CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

PhD Thesis

Dynamic Hierarchical Bandwidth Reservations for Switched Ethernet

Zahid Iqbal

CISTER-TR-181004

Dynamic Hierarchical Bandwidth Reservations for Switched Ethernet

Zahid Iqbal

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Meeting system wide timeliness requirements within Cyber-Physical Systems (CPS) is a challengingtask due to their typically complex networking infrastructure among other factors. Currentcommunication technologies do not overcome this challenge, particularly when allowing adaptationof the system for efficient bandwidth usage. A component-based design approach can helpcoping with the network complexity by allowing composition of complex applications through theintegration of independently developed adaptive components while maintaining their individualproperties. In this context, network reservations are an important design element that favors composabilityin the time domain with online adaptation by providing temporal isolation. Based onthese principles, we propose in this work a framework for supporting composability in Ethernetnetworks using ordinary COTS switches and the FTT-SE protocol. We dedicate particular attentionto the worst-case response time analysis of messages transmitted within reservations. Thisanalysis is a key element for guaranteed timeliness in an adaptive framework.In the first part of our work, we develop a new worst-case network delay analysis for sporadicreservations associated with asynchronous messages, which we call flat reservations, and assessits efficiency through extensive simulation. Our results show that our analysis is accurate, with anexact match for a significant percentage of messages in the message sets (up to 60% on average).Moreover, we were able to identify the regions in the system configuration where our analysis isaccurate, thus providing a system designer with an indication of confidence in our analysis.When multiple applications co-exist in the system, flat reservations are not adequate to providethe desired level of isolation between different applications and to meet their timing requirements.For this reason, we resort to the Hierarchical Scheduling Framework (HSF), an importanttechnique to achieve composability, particularly in the time domain as it allows reserving and partitioningthe resources in multiple levels. Hence, in the second part of our work, we implementan HSF that enforces temporal properties of the partitions, using different reservation schedulingpolicies, namely polling and sporadic servers. Our results highlight the strong partitioningcapabilities of our approach, with full temporal isolation across different hierarchical partitions.Finally, in the third part of our work, we provide a novel method to generate server interfacesthat minimizes the servers bandwidth requirement. We validate the approach with extensivesimulations using random message sets and hierarchies.

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Dynamic Hierarchical Bandwidth Reservations for Switched Ethernet

Zahid Iqbal

Programa Doutoral em Engenharia Electrotécnica e de Computadores

Supervisor: Prof. Dr. Luís Miguel Pinho de Almeida

Co-advisor: Prof. Dr. Moris Habib Yasi Behnam

May 15, 2018

Dynamic Hierarchical Bandwidth Reservations for Switched Ethernet

Zahid Iqbal

Programa Doutoral em Engenharia Electrotécnica e de Computadores

Dissertation submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Electrical and Computer Engineering at the Faculty of Engineering, University of Porto

Approved by:

President: Prof. Doutor José Alfredo Ribeiro da Silva Matos
Referee: Prof. Doutor Giorgio C. Buttazo
Referee: Prof. Doutor Reinder J. Bril
Co-advisor: Prof. Doutor Moris Habib Yasi Behnam
Referee: Prof. Doutor Paulo Bacelar Reis Pedreiras
Referee: Prof. Doutor Mário Jorge Rodrigues de Sousa
Supervisor: Prof. Doutor. Luís Miguel Pinho de Almeida

May 15, 2018

Abstract

Meeting system wide timeliness requirements within Cyber-Physical Systems (CPS) is a challenging task due to their typically complex networking infrastructure among other factors. Current communication technologies do not overcome this challenge, particularly when allowing adaptation of the system for efficient bandwidth usage. A component-based design approach can help coping with the network complexity by allowing composition of complex applications through the integration of independently developed adaptive components while maintaining their individual properties. In this context, network reservations are an important design element that favors composability in the time domain with online adaptation by providing temporal isolation. Based on these principles, we propose in this work a framework for supporting composability in Ethernet networks using ordinary COTS switches and the FTT-SE protocol. We dedicate particular attention to the worst-case response time analysis of messages transmitted within reservations. This analysis is a key element for guaranteed timeliness in an adaptive framework.

In the first part of our work, we develop a new worst-case network delay analysis for sporadic reservations associated with asynchronous messages, which we call flat reservations, and assess its efficiency through extensive simulation. Our results show that our analysis is accurate, with an exact match for a significant percentage of messages in the message sets (up to 60% on average). Moreover, we were able to identify the regions in the system configuration where our analysis is accurate, thus providing a system designer with an indication of confidence in our analysis.

When multiple applications co-exist in the system, flat reservations are not adequate to provide the desired level of isolation between different applications and to meet their timing requirements. For this reason, we resort to the Hierarchical Scheduling Framework (HSF), an important technique to achieve composability, particularly in the time domain as it allows reserving and partitioning the resources in multiple levels. Hence, in the second part of our work, we implement an HSF that enforces temporal properties of the partitions, using different reservation scheduling policies, namely polling and sporadic servers. Our results highlight the strong partitioning capabilities of our approach, with full temporal isolation across different hierarchical partitions.

Finally, in the third part of our work, we provide a novel method to generate server interfaces that minimizes the servers bandwidth requirement. We validate the approach with extensive simulations using random message sets and hierarchies.

Resumo

Satisfazer requisitos temporais ao nível de sistema é uma tarefa desafiante no contexto de Sistemas Ciber-Fisicos (CPS - Cyber-Physical Systems), que se deve, entre outros fatores, à sua infraestrutura de rede tipicamente complexa. As tecnologias de comunicação atuais não resolvem este desafio, particularmente quando se permite uma adaptação dinâmica do sistema para utilização eficiente da largura de banda. Uma abordagem baseada em componentes pode ajudar a lidar com a complexidade da rede ao permitir a composição de aplicações complexas através da integração de componentes desenvolvidos independentemente, mantendo as suas propriedades individuais. Neste contexto, as reservas de rede são um elemento de projeto importante que favorece a composabilidade no domínio temporal. Baseado nestes princípios, apresentamos neste trabalho uma abordagem que suporta a composabilidade em redes Ethernet utilizando switches COTS e o protocolo FTT-SE.

Na primeira parte do nosso trabalho, desenvolvemos uma nova análise de pior caso do atraso de rede para reservas esporádicas associadas a mensagens assíncronas, que designámos por reservas planas, e avaliámos a sua eficiência através de extensas simulações. Os nossos resultados mostram que a nossa análise é precisa, com uma correspondência exata para uma percentagem significativa do conjunto de mensagens (até 60% em média). Mais ainda, fomos capazes de identificar as regiões na configuração de sistema onde a análise é precisa, fornecendo aos projetistas de sistema uma indicação de confiança na nossa análise.

Quando múltiplas aplicações co-existem no sistema, reservas planas não são adequadas para fornecer o nível de isolamento desejado entre diferentes aplicações e satisfazer os seus requisitos temporais. Por esta razão, recorremos à Técnica de Escalonamento Hierárquico (HSF – Hierarchical Scheduling Framework), uma técnica importante para alcançar composabilidade. Assim, na segunda parte do nosso trabalho, implementámos uma HSF que impõe as propriedades temporais das partições hierárquicas, usando diferentes políticas de escalonamento de reservas, nomeadamente servidores de amonstragem ("polling") e esporádicos. Os nossos resultados realçam as fortes capacidades de particionamento da nossa abordagem, com total isolamento temporal entre as diferentes partições.

Finalmente, na terceira parte do nosso trabalho, propomos um novo método para gerar interfaces de servidor que minimizam os requisitos de largura de banda que garantem o cumprimento das restrições temporais das comunicações associadas. Validámos a nossa abordagem através de extensas simulações usando conjuntos de mensagens e hierarquias de servidores aleatórios. A validação mostrou que o método que propomos para projetar servidores de amostragem ("polling") é mais eficiente do que a abordagem tradicional de atribuir aos servidores as propriedades das mensagens que devem suportar. De facto, a nossa abordagem é exata no sentido em que uma redução mínima nas propriedades do servidor leva à violação das restrições temporais das mensagens associadas.

Acknowledgements

The years spent in PhD make a memorable time of my life; and at the end of those years, looking back, I find myself enriched with memories, with personal as well as professional development. During this journey, I came across several wonderful people whom I want to thank.

My sincere gratitude goes to my thesis supervisor, Luis Almeida, whose guidance over the course of thesis, patience, and encouragement has made this work possible. His scientific advice has been invaluable to my work. I have learnt a lot from him. I thank for the continuous support in terms of funds, and for his dedication to the DaRTES lab which made it a very pleasant place to work in. I am grateful, also, to my thesis co-advisor, Moris Behnam, in particular, for close initial interaction which helped to define the direction of our research, and for guidance to approach the FTT-SE code through flow charts, and for reviews, and helpful feedback. The visit at MDH allowed us to conclude an important part of the work, for which I would also thank Mohammad for useful discussions. I enjoyed this time working and meeting other people, in particular, my lab-mates Nabar, Jakob and Asha.

Next, I am thankful to Ricardo, the main author of FTT-SE. During the initial years, mainly, discussions with Ricardo greatly improved my understanding of the parts of the code, in particular, scheduling of the downlinks and the concept of blocks in the implementation.

I am grateful, also, to the members of my jury who gave their consent to analyse my work. They provided valuable comments and suggestions; partly, addressed in the final document.

I would like to thank all my colleagues at DaRTES who made life at FEUP enjoyable. There are a number of people to mention; recounting from the early days in FEUP, Pedro, Luis, Shuai, then, João Reis, Paulo Amaral, Ana Rita, Julio, Luis & Daniel, and more recently, Inés, Sydney, Hassan, Diana, Carlos, João, Moses and Aqsa. I have to thank Luís Oliveira for being always available when I needed to discuss something, André for his help on FEUP grid, and acquainting me with certain issues thereof, Luis Pinto, regarding different pointer-related issues in C programming that helped me in cleaning some implementation bugs. I feel lucky to have known such nice people. I thank Rui Carvalho and Bharat, close workplace neighbours, for their kindness.

I would like to thank administrative staff at FEUP and IT for their help in solving different practical issues.

Among other people, I have to thank Cihan Deniz, a compassionate friend who has always inspired and encouraged me, Ali Akca, for being a good friend and pleasant coffee company ever since first years in Portugal, and some others who have moved away since 2016, including Muhammed Ali, Selcuk Kaya, Yunus, and Yasin. I wish you the best in your future and hope that we may cross paths. I thank Anis-ur-Rehman for his kindness, help and suggestions, in particular, around various bureaucratic matters. I take this opportunity to thank other friends and housemates for their support and generosity, Mushtaq Raza, Kashif Mushtaq, Zeeshan, Hamza, Arsalan, Ajmal, Syed Aftab Rashid, Ali Awan, Asif, Niaz Ali, Bilal, Anwaar, Saqlain, Saad Sultan, Raid, Tallat, Saad Hassan, Shahkar, Hazem, Azaza, Musa, Abdul Razaq, Hassan and Nasser Alaraimi. Thank you !!

Lastly, I owe a deep gratitude to my family, my mother for her absolute love, my brothers and sister, for their love, support, and patience, over the years during my PhD, especially, my eldest brother, Muhammad Afzal.

Above all, I am thankful to God for His guidance and blessings.

The work in this thesis was supported by the grants (SFRH/BD/89731/2012) from Fundação para a Ciência e a Tecnologia (FCT), (BIM/N^{o}20/2017 – B00308) and (BIM/N^{o}42/2017 - B00308) from Instituto de Telecomunicações (IT).

Zahid Iqbal

I dedicate this work to my parents

Contents

Abstract i

Acknowledgements v

Acronyms and Abbreviations xvii

1 Introduction 1
- 1.1 System Characteristics . . . 2
  - 1.1.1 The real-time nature of the underlying system . . . 2
  - 1.1.2 Heterogeneity in applications and requirements . . . 2
  - 1.1.3 Complexity challenge: rationale for using component based design . . . 3
  - 1.1.4 The need to optimise the resource usage . . . 3
- 1.2 Motivation . . . 4
  - 1.2.1 Real-time service at the network layer . . . 4
  - 1.2.2 Real-time networks . . . 5
  - 1.2.3 Bandwidth reservation . . . 6
- 1.3 Defining the Problem . . . 7
  - 1.3.1 Hierarchical scheduling and its vision . . . 7
  - 1.3.2 Proposed solution . . . 8
- 1.4 Objectives . . . 10
- 1.5 Thesis Outline . . . 11
- 1.6 List of Publications . . . 12

2 Ethernet in Embedded Systems: the Automotive Case 15
- 2.1 Challenges of System Integration within an Automobile . . . 15
- 2.2 Ethernet-based in-Car Communications . . . 17
- 2.3 Ethernet . . . 18
  - 2.3.1 Probabilistic nature of Ethernet transmissions . . . 18
  - 2.3.2 Rationale for minimum Ethernet frame length . . . 19
- 2.4 Switched Ethernet . . . 19
- 2.5 AFDX - Avionics Full Duplex Switched Ethernet . . . 20
- 2.6 AVB - Audio Video Bridging Standard . . . 21
  - 2.6.1 An AVB system . . . 22
  - 2.6.2 Importance of synchronization . . . 22
  - 2.6.3 Stream reservation protocol (SRP) . . . 24
  - 2.6.4 Traffic scheduling in AVB . . . 25
  - 2.6.5 Schedulability analysis for AVB . . . 27
- 2.7 Time Sensitive Networking (TSN) . . . 28

2.8 Time-Triggered Ethernet (TTEthernet) . . . 30
2.8.1 Architecture model . . . 30
2.8.2 Traffic scheduling . . . 31
2.9 FTT-SE: a Brief Overview . . . 33
2.9.1 Handling synchronous & asynchronous traffic . . . 34
2.9.2 Building traffic schedules . . . 34
2.10 A Qualitative Comparison of Different Technologies . . . 34
2.11 Summary . . . 41

3 Traffic Scheduling Concepts **45**
3.1 Server-based Scheduling . . . 45
3.1.1 Polling server . . . 46
3.1.2 Deferrable server . . . 46
3.1.3 Sporadic server . . . 48
3.2 Hierarchical Scheduling . . . 50
3.3 Guaranteeing Quality of Service . . . 53
3.3.1 QoS at network layer . . . 54
3.3.2 Scheduling and QoS in Ethernet . . . 56
3.3.3 Scheduling in FTT-SE . . . 57
3.4 Summary . . . 58

4 Analyzing the Efficiency of Sporadic Reservations on Ethernet with FTT-SE **61**
4.1 Flat Servers within FTT-SE . . . 62
4.2 System Model . . . 64
4.2.1 Network model . . . 64
4.2.2 Traffic model . . . 65
4.2.3 Response time analysis . . . 65
4.3 Evaluation . . . 68
4.3.1 System setup . . . 68
4.3.2 Experiments . . . 69
4.4 Lessons Learnt . . . 79
4.5 Summary . . . 81

5 Supporting Hierarchical Reservations within FTT-SE using Polling Servers **85**
5.1 Hierarchical Scheduling Framework in FTT-SE . . . 86
5.1.1 Servers integration within FTT-SE . . . 87
5.1.2 Servers and streams model . . . 88
5.1.3 Scheduling model and execution . . . 91
5.2 Schedulability Analysis . . . 93
5.3 Evaluation . . . 97
5.3.1 Experimental setup . . . 97
5.3.2 Analysis results vs. observation . . . 98
5.3.3 Checking temporal isolation . . . 98
5.3.4 Verifying temporal isolation with random simulations . . . 102
5.4 Summary . . . 103

6 Supporting Hierarchical Reservations within FTT-SE using Sporadic Servers 109
6.1 Implementing Hierarchical Sporadic Servers . . . 109
6.1.1 Scheduling algorithm . . . 110
6.1.2 Replenishment management . . . 111
6.1.3 Processing message arrivals . . . 115
6.1.4 Handling message packets . . . 115
6.2 Evaluation . . . 116
6.2.1 Experimental setup . . . 116
6.2.2 Experiments . . . 117
6.3 Summary . . . 119

7 Design of Reservations 121
7.1 Server Design for Hierarchical Polling Servers . . . 121
7.1.1 Modified system model . . . 122
7.1.2 Generating server interfaces . . . 123
7.2 Evaluation . . . 128
7.2.1 The worst-case behavior . . . 128
7.2.2 The average case behavior . . . 129
7.2.3 Root server utilization . . . 130
7.3 Summary . . . 131

8 Experimenting with Hierarchical Reservations on FTT-SE 133
8.1 Components of the Experimental Framework . . . 133
8.1.1 Structure of hierarchies . . . 134
8.1.2 Generating application message set . . . 135
8.1.3 Filling in server parameters . . . 136
8.1.4 Repository of servers . . . 136
8.1.5 Message activations . . . 137
8.2 Application Design and Execution Flow . . . 139
8.3 Summary . . . 140

9 Conclusion and Future Work 143
9.1 Thesis validation . . . 144
9.2 Future Work . . . 146

List of Figures

1.1 Illustrating bandwidth reservations 6

2.1 Half duplex Ethernet 19
2.2 Typical switch internal architecture 20
2.3 sub-VL scheduling in ADFX 22
2.4 Synchronization mechanism and delay calculation 23
2.5 Attribute propagation through the network 25
2.6 An Ethernet frame with IEEE 802.1Q VLAN tag 25
2.7 *talker advertise* and *listener ready* propagation 26
2.8 An example of CBSA operation 27
2.9 An example of time-aware shaper 29
2.10 Time-aware shaper, guardband before critical transmissions 29
2.11 Time-aware shaper, preemption approach 30
2.12 TTEthernet cluster example 31
2.13 TTEthernet Rate constrained (RC) traffic 33
2.14 The FTT-SE EC structure 33
2.15 Downlink scheduling in FTT-SE 35
2.16 Determininsm and QoS in Ethernet-based technologies 40

3.1 Background service 47
3.2 Polling service 47
3.3 An example of a high priority Deferrable Server 48
3.4 An example of Sporadic Server operation 49
3.5 Critical instant of the system with sporadic server 50
3.6 A complex scenario of message scheduling with sporadic server (1) 51
3.7 A complex scenario of message scheduling with sporadic server (2) 52

4.1 Flat servers within FTT-SE 62
4.2 Main system components 63
4.3 Operation of the shapers 64
4.4 Illustrating the sources of interference. 66
4.5 Impact of inserted idle time. 66
4.6 Experimental method 71
4.7 Comparing analytic estimates and observed values of RT 72
4.8 Histogram of the percentage of messages per data set for each category 73
4.9 Reduced simulation trace 75
4.10 Harmonic vs primes, pc of matches between analysis & observation 75
4.11 Harmonic vs primes, max pc increase of RT over RT_o 76
4.12 Harmonic vs primes, pc increase of RT over RT_o over all messages 76

4.13 Different period ranges, pc of matches between analysis & observation 77
4.14 Different period ranges, max. pc increase of RT over RT_o 78
4.15 Different period ranges, pc increase of RT over RT_o over all messages 79
4.16 Different values of LW, pc of matches between analysis & observation 80
4.17 Different values of LW, max. pc increase of RT over RT_o 81
4.18 Different values of LW, pc increase of RT over RT_o over all messages 82
4.19 Varying link utilization, pc of matches between analysis & observation 83
4.20 Varying link utilization, max. pc increase of RT over RT_o 83
4.21 Varying link utilization, pc increase of RT over RT_o over all messages 83

5.1 An example server hierarchy 86
5.2 Partitioning the available bandwidth at different levels 87
5.3 Hierarchical Server Based Scheduling (HSS) architecture 88
5.4 Flat and hierarchical reservations 90
5.5 Message scheduling with flat and hierarchical reservations 92
5.6 The scheduling model 93
5.7 The supply bound function assuming the polling server. 94
5.8 The experimental setup, with three slaves and the master node. 97
5.9 Independent Server Hierarchies (ISH) prepared at each source station 98
5.10 Message response times with periodic arrival patterns 99
5.11 Response times of messages m_4 and m_5 with bursty m_5 activations 102
5.12 The experiment setup for random simulations 104
5.13 Message response times with *regular* mode 106
5.14 Message response times with *induced congestion* mode 107

6.1 Example of scheduling messages with a sporadic HSF 112
6.2 Independent Server Hierarchies (ISH) in each source station 117
6.3 Response times of messages m_4 and m_5 with bursty m_5 activations 118
6.4 Messages in station B are protected from bursts in m_{11} and m_{13} 120

7.1 An example hierarchy with six servers and four message streams 122
7.2 Different candidate interfaces for Srv_{28} to schedule AS_{53} 126
7.3 Interface composition with rational approach 127
7.4 Minimum time to the deadline for the message sets 129
7.5 Interface composition with naive approach 130
7.6 Distribution of average response time as percentage of deadline 131
7.7 Total utilization of the root servers per message set 132

8.1 FTT-SE internal layering (left) and experimental platform (right) 134
8.2 Adjacency list represenation for an ISH 135
8.3 Application design and execution flow with HSF within FTT-SE 140

List of Tables

2.1 Comparison of different technologies 43

3.1 A task system with two periodic tasks and a polling server 46
3.2 A task system with two periodic tasks and a deferrable server 48
3.3 A task system with two tasks and a sporadic server 50

4.1 Interference set of message m_{43} 74
4.2 Experiment utilization values 79

5.1 Server parameters for stations 100
5.2 Specification of message parameters 100
5.3 Messages measured and calculated response times. 101
5.4 Messages set parameters 104
5.5 Servers' parameters 105

7.1 Message Parameters 126
7.2 Interface candidates for leaf servers 127

Acronyms and Abbreviations

ABS	Antilock Braking System
ADAS	Advanced Driver Assistance Systems
ADN	Aircraft Data Networks
AFDX	Avionics Full-Duplex Switched Ethernet
ASIP	Application Specific Instruction-set Processor
AVB	Audio Video Bridging Standard / Systems
BAG	Bandwidth Allocation Gap
BE	Best-Effort
CAN	Controller Area Network
CBD	Component-based Development
CBS	Constant Bandwidth Server
CBSA	Credit Based Shaping Algorithm
COTS	Commercial Off-the-Shelf
CPS	Cyber Physical Systems
CSMA / CD	Carrie Sense Multiple Access with Collision Detection
DES	Distributed Embedded Systems
DGS	Distributed Global Scheduling
DiffServ	Differentiated services
DM	Deadline Monotonic
DS	Deferrable Server
DSS	Dynamic Sporadic Server
EC	Elementary Cycle
ECU	Electronic Control Units
EDF	Earliest Deadline First
EDP	Explicit deadline periodic
ES	End System
ESC	Electronic Stability Control
ET	Event-Triggered
FCFS	First come, first served
FIFO	First in, first out
FLOWSPEC	Flow specification
FPS	Fixed Priority Scheduling
FQTSS	Forwarding and Queuing Enhancements for Time-Sensitive Streams
FTT-SE	Flexible Time-Triggered Switched Ethernet Protocol
GARA	Globus Architecture for Reservation and Allocation
GCD	Greatest common divisor
HaRTES	Hard Real-Time Ethernet Switching

HDTV	High-definition television
HSF	Hierarchical Scheduling Framework
HSS	Hierarchical Server-based Scheduling
IP	Internet Protocol
ISH	Independent Server Hierarchy
LCM	Least common multiple
LDAP	Lightweight Directory Access Protocol
LEC	Length of EC
LIN	Local Interconnect Network
LW	Length of Asynchronous Window
MOST	Media Oriented Systems Transport
MPLS	Multiprotocol Label Switching
MRP	Multiple Registration Protocol
MTU	Maximum Transmission Unit
NIC	Ethernet Network Interface Card
NoC	Network-on-Chip
NRT	Non Real-Time
NS	Network Switch
OEM	Original Equipment Manufacturer
OMNeT++	Objective Modular Network Testbed in C++
PCP	Priority Code Point
PROFINET	Process Field Net
PROFINET-IRT	PROFINET-Isochronous Real-Time
PS	Polling Server
PTP	Precision Time Protocol
QoS	Quality of Service
Q-RAM	QoS based Resource Allocation Model
rbf	Request bound function
RBS	Reduced Buffering Scheme
RC	Rate-Constrained
RM	Rate Monotonic
RMS	Rate Monotonic Scheduling
RSVP	Resource Reservation Protocol
RT	Response time
sbf	Supply bound function
SFD	Store-and-forward delay
SRP	Stream Reservation Protocol
SS	Sporadic Server
ST	Scheduled Traffic
TAS	Time Aware Shaper
TBS	Total Bandwidth Server
TM	Trigger Message
TSN	Time Sensitive Networking
TT	Time-Triggered
TTEthernet	Time-Triggered Ethernet
TTSoC	Time-Triggered System-on-a-Chip
VL	Virtual Link

Chapter 1

Introduction

The advancement in technology has greatly impacted the lives of people. The digital revolution that marked the beginning of the information age allowed a widespread production and use of digital logic circuits and its derived technologies such as computers and cell phones. The microprocessor underlies this revolution, and with its steadily increasing power and shrinking dimensions, it has enabled computer technology to be embedded into a wide range of objects of everyday use. Examples include devices such as mobile phones, in-car navigation systems, portable music players, telephone answering machine, robot controllers etc. Such systems use a computer system inside a larger system to provide control and computation functions and are referred to as *embedded systems* [1]. These systems have an ongoing interaction with a dynamic external environment and are characterised by executing a predefined dedicated function.

Driven by demands for the economy of scale, and for simpler and more efficient solutions, there has been a continuous evolution in the technology for embedded systems. This led to declining costs of technology as well as an increase in the computing power. Moreover, to realise the objective of certain systems, connectivity was provided between multiple processors giving rise to *Distributed Embedded Systems* (DES). These factors coupled with a widespread network infrastructure enabled to harness the benefits of technology in unprecedented ways [1, 2]. Embedded systems thus find their applications in many areas such as consumer electronics, industrial automation, automotive and avionics industry, biomedical engineering etc.

Distributed embedded systems help realise the objective of *resource sharing* among distributed components. It is possible to deploy distributed protocols over a network allowing different network elements to coordinate their activities and share their data with each other to accomplish a single task or a set of tasks. An example of such distributed coordination is a product assembly line in industrial automation or a brake-by-wire system within cars.

Adding another dimension of complexity, we have systems that interact with a dynamic external environment and combine models of both environment and computing platforms, known as *Cyber-Physical Systems* (CPS), to improve their performance. For CPS to be open and flexible, e.g., allowing applications to enter or leave the system or the system topology to change, we need to provide resource efficiency in varying operational scenarios. The interconnection network is an

important element in this respect. The networking medium is a shared resource, and there have been tremendous efforts to devise strategies to access this medium efficiently. The network plays a central role in supporting system-wide properties. However, current communication technologies do not fully meet the communication requirements of CPS, particularly concerning timeliness and reliability together with scalability, flexibility, and openness.
To this end, we postulate that the resource reservation paradigm is an adequate means to cater for scalable, open and adaptive latency constrained communications towards efficient CPS. This dissertation will strive to explore network resource allocation and scheduling strategies that, in particular, allow applications with different levels of criticality, possibly with a non-continuous operation, to share the network medium in an efficient manner.

1.1 System Characteristics

CPS integrate computation, networking, and physical processes. They encompass a broad scope in which the system is not only real-time, networked and distributed but can be adaptive, predictive as well as intelligent. They may also require security against malicious attacks, and intrusion detection mechanisms depending on the application area. CPS thus need improved design tools and design methodologies that can ease the development process, handle complexity, and in particular support validation and verification. For example, the work in [3, 4] lists the challenges related to CPS development whereas the work in [5] gives a concept map for CPS and lists several projects currently addressing CPS related development.
In our work, however, we limit our focus to designing efficient bandwidth reservation mechanisms for CPS. From this perspective, we list certain system characteristics that we must take into account in the development of the work.

1.1.1 The real-time nature of the underlying system

CPS, due to their interaction with a dynamic environment, are necessarily *real-time systems* for which correct operation depends on the logical result of the computation and frequently on the time of the output delivery. In fact, failure to respond in time can be as bad as a wrong response. The concept of the *deadline* governs the notion of real-time. For the system outputs to be acceptable, these have to be generated by the deadline. The severity of the consequences that may result from a deadline miss classifies the real-time systems as *hard real-time* systems and *soft real-time* systems [6]. We consider CPS that impose strict requirements on their timing behaviour and require that such requirements are met.

1.1.2 Heterogeneity in applications and requirements

A CPS intrinsically will consist of multiple applications, concurrently accessing a shared network resource. These applications may exhibit different timing constraints and resource access patterns. Efficient resource allocation is thus an important issue. It is necessary that disparate

applications complete their computations within their timing constraints and do not impact the real-time behaviour of other applications that use the same resources. More specifically, a misbehaving application may miss its deadline but should not affect other applications. This property is known as *temporal isolation*.
To reliably run combinations of such applications and effectively manage the resource is a challenging task. System performance is thus greatly affected by the resource allocation strategy. Allocating the resource at a high rate may be inefficient whereas allocating at a lower rate may result in long response times which may degrade service provided by the applications or render it meaningless altogether.

1.1.3 Complexity challenge: rationale for using component based design

CPS rely, up to a large extent, on networking infrastructures, frequently large ones. These necessarily play a central role in supporting the needed system-wide properties, being timeliness a particularly important one as dictated by the dynamics of the associated physical processes. Owing to their complexity, large-scale CPS are hard to assess through a single holistic view. The traditional approach addresses the challenge of complexity by partitioning the system functionality across several components and building them independently. Such an approach, however, entails a standard design and development process so that different system parts can be seamlessly integrated. Otherwise, on each system integration, additional development efforts need to be invested in making system parts compatible, thus, increasing development costs and time-to-market as well as quality risks. The traditional practice in the industry has been to build replaceable system parts meeting pre-defined standards and then integrating them to make systems. Such paradigm is known as *Component-based Development* (CBD). A component is a self-contained system part that can be developed and tested in isolation and can be composed with the rest of the system through well-defined interfaces [7]. Component-based development can, thus, address the complexity of large-scale CPS by allowing a dependable composition of the system parts [8]. Therefore, we are also interested in modelling the communications in a composable way, i.e., knowing the temporal properties of the components, we want to verify that the system as a whole or parts preserve their timeliness properties when different components interact in different possible manners. Thus, we need methods to ensure timing guarantees during the integration and reuse of the system components.

1.1.4 The need to optimise the resource usage

Furthermore, CPS integrate with the physical processes through sensors and actuators to achieve their functional objectives. Thus CPS have to be reactive and timely which imposes stringent requirements on the usage of resources (in this case network bandwidth). With efficient use of resources, more applications can be accommodated while meeting their requirements. This characteristic relates closely to the one in Section 1.1.2, and essentially can be read as optimizing resource usage in the presence of multiple heterogeneous applications. Particular constraints in

most embedded systems (that will make part of CPS but nonetheless maintain their non-functional requirements) dictate efficiency of resource usage as a foremost concern. A resource allocation strategy aiming in this direction results in savings such as power, i.e., over-provisioning resources may result in undue power consumptions. For such systems, a capability to tailor the resource usage according to the needs of each application is a desirable feature [9].

1.2 Motivation

This dissertation focuses on the efficient management of the network bandwidth to support complex dynamic network transactions with timeliness guarantees.

1.2.1 Real-time service at the network layer

Considering latency constrained applications, we find that common services at the network layer are not adequate. In most data communications, reliable delivery of data is the desired goal. Examples include email, web etc. Real-time communications further require that data arrives in a timely manner. However, FIFO queues within network switches and routers implement a best effort service model. With this model, all packets receive the same quality of service. Under light load conditions, such quality is good; however, applications receive poor service when the network is heavily loaded. To this end, the works in [10, 11] explored more efficient non-FIFO packet scheduling algorithms and the idea of allocating resources selectively. They support quality of service (QoS) by having an admission control mechanism and defining adequate QoS metrics such as the bound on maximum packet delay. With this model, applications are serviced along two different QoS dimensions namely fidelity and latency. However, we observe that, if an application needs low latency, there is a significant probability that it will need to tolerate some lost packets. Applications that require high fidelity, on the other hand, may experience long response times. Both of these situations are not suitable for applications that need hard real-time guarantees.
Similarly, using the Internet Protocol (IP), performance is not guaranteed; packets can be lost, delayed, reordered, corrupted, and the task of rearranging and recovering is left to the protocols in the upper layers. The service at the network layer, by itself, is insufficient; it just tries to deliver the packets. Real-time applications often do not work well across the Internet because of variable queueing delays and congestion losses. Traffic passes through many hops that are maintained by different ISPs.
There have been some works that strive to support soft real-time applications using IP. Some architectures were proposed that provide QoS such as Integrated Services (IntServ) and Differentiated Services (DiffServ) [12] and traffic engineering solutions such as Multi-protocol Label Switching (MPLS) [13]. However, in spite of these solutions, it has been challenging to support soft real-time applications over the Internet.

1.2.2 Real-time networks

On the other hand, many networks capable of offering bounded latency, so-called real-time networks, have been developed throughout the years. In most cases, these focused on the strictness of timing guarantees with hardly any support to on-line adaptation as required for efficiency reasons and to support new trends towards open and evolvable systems.
In the industrial automation realm, these real-time networks have been known as field-buses, and there have been many different technologies such as CAN [14, 15, 16], LIN [17, 18], FlexRay [19, 20].

1.2.2.1 On using Ethernet with QoS guarantees

The simplification of planning, deployment and maintenance as well as a reduction of costs pushed towards more open networks and using flexible technologies such as general purpose Ethernet. Ethernet has been entering the industrial automation domain and in distributed embedded systems and many variants appeared to cater for real-time requirements. Following this trend, several attempts were made to reconcile the flexibility and openness of Ethernet with the provision of deterministic QoS guarantees.
The automotive sector, in particular, has been devoting considerable attention to this combination [21]. One approach that is attracting significant attention and became standardised recently is IEEE 802.1BA: Audio Video Bridging Systems - AVB [22]. This solution is based on resource reservation and subsequent enforcement of those reservations. However, it requires special switches capable of handling resource reservation requests and doing traffic shaping, and AVB switches are still almost non-existent in the market. Although the resource reservation may be done dynamically, each reservation is for a fixed value (e.g. bandwidth), with remaining capacity used only for background traffic.
Other technologies such as AFDX [23], PROFINET-IRT [24] and TTEthernet [25], have been used in static scenarios with well-defined requirements, as typically found in distributed embedded systems and industrial systems in general. These protocols can offer some limited forms of reservations for specific traffic types. For example, time-triggered frameworks allow reserving fixed windows or slots for the transmission of different kinds of traffic, both periodic and aperiodic traffic. The channel bandwidth is available on these slots or windows on a periodic basis. The bandwidth allocation is exclusive, which means that if no traffic of the respective type is pending, the bandwidth cannot be used by other types of traffic and is, thus, wasted. Moreover, fixed slots within rigid cyclic frameworks impose a compromise between bandwidth and response time, i.e., low response times can only be achieved with high bandwidth requirements. Within AFDX [23], end stations communicate through virtual links (VL) where a certain bandwidth is allocated. When an end station has multiple applications, these are scheduled by Round-robin policy for using the VL. This scheduling treats all applications equally and therefore can negatively impact an application's real-time behaviour.

1.2.3 Bandwidth reservation

To introduce the concept of bandwidth reservation, we consider the case of soft real-time applications that can tolerate some deadline misses. Traditionally, such applications were scheduled in the background after higher priority traffic was scheduled thus making their response times very long or causing deadline misses especially in cases where the system was overloaded with higher priority traffic. Providing bandwidth reservation for such traffic types with certain timing may result in them being more responsive and getting good service. Hierarchical reservations, on the other hand, realise the objective of composing a complex system with many applications with sub-parts that need to share a common resource. Imagine an application that has some flows with hard real-time requirements and others with soft real-time requirements. We can create one reservation for the application and then we can subdivide this reservation into two reservations, where a larger bandwidth can be allocated for the hard real-time flows and a smaller reservation for the soft real-time streams. Moreover, the scheduler at the application level can assign a higher priority to the reservation for the hard real-time flows (Figure 1.1). In this way, different flows also receive a differentiated service as well as isolation in the temporal domain. Similarly, if there are more applications in the system, these will be scheduled by a system level scheduler. Finally, allowing dynamic reservations leads to efficient bandwidth utilisation. As a simple example, we compare with a static scenario where bandwidth would be wasted if the allocated flows were no longer available. With dynamic reservations, bandwidth can be allocated on request, and as well the reservations can be terminated if an application leaves the system thereby leading to more open and flexible systems. The aforementioned real-time protocols lack the flexibility to allow either arbitrary reservation scheduling policies with a hierarchical composition to support complex applications or the servers dynamic management for the sake of efficiency.

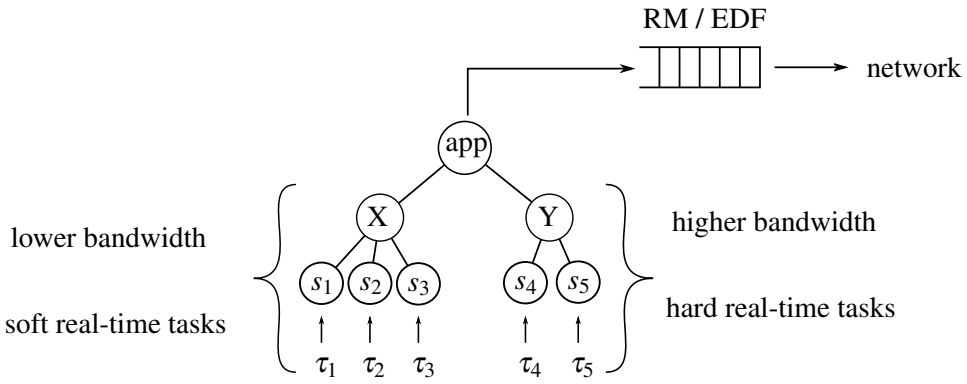


Figure 1.1: Illustrating bandwidth reservations; assigning different shares to different flows. Scheduling enforces the allocated bandwidth to each flow $\{\tau_1, \cdots, \tau_5\}$.

1.3 Defining the Problem

Supporting multiple applications or applications with multiple components that access a single shared resource and need timeliness guarantees is a challenging task. To understand this complexity, let us consider CPU time to be a shared resource and a single scheduling strategy for different applications. Even with specific priority assignments such as Rate Monotonic (RM) or Earliest Deadline First (EDF), the system cannot provide isolation between different applications and guarantee the level of service that applications will receive for cases where applications misbehave; i.e., do not respect their resource utilisation specifications. For example, a bursty high priority application can starve a lower priority application of the CPU time, or if a particular application job overruns its execution time, it may cause another application to miss its deadline. This issue of simple priority-based scheduling being not adequate for application QoS guarantees in complex systems has been stated in [11].
Reservation-based scheduling, also known as *server-based scheduling*, can solve this problem. With this scheduling strategy, Q time units are guaranteed on CPU every P time units. This abstraction, known as *server*, helps providing temporal isolation between different applications. In this way, a misbehaving application cannot negatively impact other applications.

1.3.1 Hierarchical scheduling and its vision

Traditionally, low priority aperiodic requests have been served in the remaining time after the hard real-time periodic applications were scheduled. Server-based scheduling techniques have been used to improve the response time of aperiodic requests without jeopardising the schedulability of periodic tasks. To this end, the work in [26] proposed the Polling Server (PS) and Deferrable Server (DS) policies and the work in [27] proposed the Sporadic Server (SS) policy. These methods differ in the way reserves are replenished. These methods used static priorities, e.g. Rate Monotonic (RM) for periodic tasks. Arguing against the low CPU utilisation of RM policy, Spuri and Buttazzo [28] proposed new server algorithms under dynamic priorities, namely Earliest Deadline First (EDF) for periodic tasks. They introduced a set of new algorithms such as Dynamic Priority Exchange, Dynamic SS (DSS) and Total Bandwidth Server (TBS). The work in [29, 30] supports soft real-time multimedia streaming through a reservation policy known as Constant Bandwidth Server (CBS). When the budget is exhausted, the CBS principle allows increasing its deadline and replenishes the budget immediately. DSS, on the other hand, becomes idle until next replenishment when its budget is exhausted. Thus, CBS improves performance offered to soft real-time tasks. In a simulation experiment which computes mean tardiness over all instances of a soft task, CBS performs significantly better (achieves smaller tardiness of the task set) in comparison to DSS. These algorithms varied concerning efficiency and implementation overheads. All these works help designers of hard real-time systems to select the best reservation technique according to their needs.
A complex application with sub-parts (components) can be allocated a certain bandwidth (parent reservation). Then, this reservation can be divided to create child reservations that can be assigned

to the sub-parts of the application. This division can continue through several levels if the sub-parts have further components. This approach helps to deploy the component-based design at run-time. It brings the following benefits:

- The system bandwidth is subdivided into reservations to run multiple applications.
- These reservations are scheduled by a system level scheduler.
- The reservations can be subdivided into smaller reservations that execute parts /components of applications.
- By allocating applications to their reservations, we achieve temporal isolation.
- The reservations can be adapted on-line according to effective needs, reducing bandwidth waste.
- A scheduler manages the (sub)reservations at each level. We can use a schedule that best fits the requirements of the components to be scheduled at that level.
- Inside a reservation, each component can be analysed. This analysis can be independent of other elements in the system. The component interface abstracts its resource requirements.
- The system level analysis can be carried out compositionally by considering component demands through their interface.

1.3.2 Proposed solution

A component-based design approach can help coping with the network complexity by allowing composition of complex applications through the integration of independently developed components while maintaining their individual properties. In this context, network reservations are an important design element that favour composability in the time domain by providing *temporal isolation*.
Our work involves research and development of efficient network reservation mechanisms in single switch Ethernet networks. Following a time-triggered approach using the Flexible Time-Triggered protocol for Switched Ethernet (FTT-SE) [31] which combines the flexibility of online traffic scheduling with the time-triggered model, we investigate the efficiency of flat reservations provided natively by the protocol and the effectiveness of implementing hierarchical server-based scheduling. The thesis supported by this dissertation is as follows:

> *The resource reservation paradigm is an effective means to segregate the communications from multiple heterogeneous applications in a distributed embedded system, potentially with mixed criticality levels, diverse real-time requirements and evolving configurations, thus supporting composability. In particular, we claim that this paradigm can be efficiently deployed over Ethernet, using the FTT-SE protocol, providing multiple levels of traffic isolation and constrained latency guarantees.*

1.3.2.1 Motivating example

We consider the example of modern car technology. The growing number of automotive applications drive up the bandwidth requirement as well as the introduction of computer-based applications and control systems, the associated electronics and wiring result in higher overall system costs. Automotive thus serves as an example case where Ethernet deployment is being considered to achieve high bandwidth and a low system cost [32].

The electronics in a car are divided into *domains* where each domain implements certain functionality [33, 32]. The powertrain is the group of components that generate energy to power the vehicle on the road. Apart from including the engine, transmission, shafts, and wheels, powertrain includes sensors and controls. Sensors measure flow, pressure, speed, torque, angle, and position of various items. To make sure that the right amount of fuel is injected into the engine, pressure sensors measure the fuel pressure, which affects the timing of the injection. Thus computer-based control is responsible for sampling sensor values within certain periods and to actuate the valves at certain frequencies. The admissible latencies of such tasks are typically in the order of microseconds.

The chassis includes the internal framework that supports the powertrain, as well as all components, such as brakes, steering, and suspension. Similar to the powertrain, the sensors and controls for the chassis domain have exact timing requirements with maximum controlled latencies.

The body domain includes such things as heating and A/C, seat controls, window controls, lights, etc. These controls and sensors typically require low bandwidth and can handle higher latency (milliseconds).

Driver assistance systems include items such as in-vehicle navigation (using GPS or similar), automatic parking, collision avoidance systems, intelligent speed adaptation or advice, etc. These systems typically have their sensors and their dedicated computers, which implement controls that often interact with other systems (such as the powertrain, chassis). The driver assistance systems often require more computing power and higher bandwidth communications to sensors, but they can typically handle latencies in the hundreds of microseconds.

A car is a safety-critical system, and we can see that there are different sources of traffic sharing the medium and having different latency and criticality requirements. Traditional Ethernet, however, does not have a bandwidth control mechanism to allocate different shares to different streams, and hence it cannot be used to transmit heterogeneous data from multiple sources safely. Without the use of a reservation mechanism, we face an integration problem, i.e. interference in the temporal domain may cause applications latency requirements to be missed.

Automotive is, therefore, an example of a system where communications can be modelled using reservations, such that applications achieve temporal isolation and meet their requirements independently of each other. Moreover, hierarchical reservations provide further benefits. For example, when composing different applications in the system, the integrator has to know, only, the high-level reservation of each application and not the sub-reservations that are internal to the applications, simplifying the composition. On the other hand, many of the automotive systems will work in certain conditions, only, and can be switched off in other. In that case, bandwidth

can be managed dynamically, reclaiming bandwidth from systems that are not operating at each moment and giving it to other systems that can take advantage of it for improved QoS. A work that investigates operational flexibility for safety-critical systems is reported in [34]. With hierarchical reservations, this management can be done at the high level that represents applications, only, and the internal allocation is left for the application logic. Both composition and dynamic bandwidth management become significantly more complex when there are flat reservations available, only.

1.4 Objectives

In this work, we aim at developing dynamic resource reservation mechanisms for CPS for the sake of efficiency, flexibility and timeliness. In distributed embedded systems, it is hard to guarantee timeliness in cases when the network depth grows, or changes occur in the system topology or its dynamics, i.e., addition or removal of system components, software updates, clock drifts etc. Such changes may negatively affect CPS applications especially those with hard real-time requirements, essentially due to mutual interference. The idea is to engineer communications for robust CPS design that can adapt to changing scenarios within bounded latency.
We propose researching and developing efficient network reservation mechanisms primarily in single switch Ethernet networks (and later extension to multi-switch networks) that support dynamic resource reservation with dynamic reconfiguration and sharing of free resource capacity without jeopardising the isolation properties of reservations. In particular, we aim to develop resource reservations with the following features:

i) application-oriented semantics with a hierarchy, i.e., a complex application could be divided into sub-applications and reservations can be associated with applications with a single high-level interface but supporting internal nested reservations as requested by the applications;

ii) flexible reservations, which provide a minimum guaranteed QoS and then offer more whenever there are available resources;

iii) scalable solution so that it can work on networks with many flows associated with many and heterogeneous applications. For example, from industrial equipment to vehicles.

iv) load-aware solution, allocating less additional resources on more loaded links, meeting end-to-end constraints, acting on deadlines, on priorities or even on the period/capacity of the reservations, according to what the underlying protocol allows.

We aim at providing a general solution that can positively impact on the design and operation of CPS, particularly automotive and remote interaction systems. In particular, we will use the following open-source Ethernet technology that is particularly adequate since it is amenable to hierarchical reservations and allows open traffic scheduling policies, namely FTT-SE [35, 31].

1.4.1 Contributions

The contributions of this thesis are the following:

I) **Analyzing the efficiency of flat sporadic reservations**
This work concerns an extensive assessment of the efficiency of a delay analytic model for asynchronous streams being scheduled within FTT-SE through flat sporadic reservations. The efficiency study was carried out with a simulation framework that compared, for large data sets, the maximum observed message response times from the protocol scheduler with the corresponding analytic delay upper bounds. By changing system configurations along different dimensions, we found out how did the analysis efficiency vary with such changes and which configurations that would favour more detailed analyses. Such study is instrumental in providing guidelines for efficient system designs. This work has been published in [36]. Parts of this work have been presented in [37], [38], and [39].

II) **Implementation of different hierarchical resource reservation policies**
This work presents a Hierarchical Scheduling Framework (HSF) with different server policies, namely polling and sporadic servers. We have extended the scheduling model within FTT-SE with an implementation of the HSF that creates virtual partitions in the time domain by dividing and subdividing the network bandwidth in a hierarchical way. Different applications can be mapped to different parts of the hierarchy. Empirical verification of the effectiveness of this approach through simulation set-up and real-system runs as well as a schedulability analysis for polling servers was presented in [40]. The work in [41] adapts server-based architecture to use the sporadic server model for improved responsiveness.

III) **Techniques to design reservations**
In this work, we investigate the server design problem. This problem relates to choosing values of the server parameters (i.e., budget and period), given a particular internal workload such that it is schedulable independently of other components in the system and uses the least bandwidth [42]. The presented approach builds on the knowledge (requested bandwidth) of the connected streams in a given hierarchy and generates interface values at the leaf servers. While generating an interface at the parent node, all the children are evaluated simultaneously concerning the available interface option for each child. This allows a parent node to schedule all the children at each of its scheduling instant, which impacts on the application response time.

1.5 Thesis Outline

The thesis is organised as follows. This introductory chapter is followed by Chapter 2 which motivates the case of Ethernet in embedded systems with automotive as an example case. This chapter also presents some Real-Time Ethernet-based technologies and illustrates our choice of using FTT-SE in this work. Chapter 3 discusses some scheduling techniques that can be used to achieve QoS within Ethernet. Chapter 4 presents an analysis technique and evaluation of native

sporadic reservations within FTT-SE. This chapter concludes by discussing some features of the message set that reduce the efficiency of the analysis. Chapter 5 presents the implementation and analysis of hierarchical reservations with polling server policy in FTT-SE, whereas Chapter 6 uses the sporadic server policy. A resource efficient design technique for hierarchical polling reservations is presented in Chapter 7. In Chapter 8, we present the implementation aspects and design concepts for developing applications based on hierarchical reservations. This chapter also presents the simulation framework that was used for evaluating the HSFs. Finally, Chapter 9 summarises the findings, validates the thesis and presents some future work directions.

1.6 List of Publications

Some of the contributions listed above or presented in this thesis have appeared in the following publications.

(Chapter 5) Z. Iqbal, L. Almeida, R. Marau, M. Behnam, and T. Nolte, "Implementing Hierarchical Scheduling on COTS Ethernet Switches Using a Master/Slave Approach," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES 2012)*, pp. 76–84, June 2012

(Chapter 6) Z. Iqbal, L. Almeida, and M. Behnam, "Implementing Virtual Channels in Ethernet using Hierarchical Sporadic Servers," in *The 12th International Workshop on Real-Time Networks (RTN 2013) in conjunction with the 25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*, July 2013

(Chapter 7) Z. Iqbal, L. Almeida, and M. Behnam, "Designing Network Servers within a Hierarchical Scheduling Framework," in *30th Annual ACM Symposium on Applied Computing (SAC 2015)*, pp. 653–658, Apr. 2015

(Chapters 4 and 6) Z. Iqbal and L. Almeida, "Towards an analysis for hierarchies of sporadic servers on Ethernet," in *11th IEEE Symposium on Industrial Embedded Systems (SIES 2016)*, pp. 1–6, May 2016

(Chapter 4) Z. Iqbal, L. Almeida, and M. Behnam, "Efficiency study for sporadic servers on Ethernet with FTT-SE," in *9th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2016) in conjunction with the 37th IEEE International Real-Time Systems Symposium (RTSS 2016)*, pp. 25–26, November 2016

(Chapter 4) Z. Iqbal, L. Almeida, and M. Ashjaei, "Analyzing the Efficiency of Sporadic Reservations on Ethernet with FTT-SE," in *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2017)*, pp. 1–8, Sept 2017

(Chapter 4) Z. Iqbal, L. Almeida, M. Ashjaei, and M. Behnam, "On the efficiency of sporadic servers on Ethernet with FTT-SE," *SIGBED Rev.*, vol. 14, pp. 32–34, Nov. 2017

1.6.1 Other publications

Publications resulting from collaboration work related with but beyond this thesis, carried out with other colleagues:

(I) M. Behnam, Z. Iqbal, P. Silva, R. Marau, L. Almeida, and P. Portugal, "Engineering and Analyzing Multi-Switch Networks with Single Point of Control," in *1st International Workshop on Worst-Case Traversal Time (WCTT'11) in conjunction with the 32nd IEEE International Real-Time Systems Symposium (RTSS'11)*, pp. 11–18, Nov. 2011

(II) R. Marau, M. Behnam, Z. Iqbal, P. Silva, L. Almeida, and P. Portugal, "Controlling Multi-Switch Networks for Prompt Reconfiguration," in *9th IEEE International Workshop on Factory Communication Systems (WFCS 2012)*, pp. 233–242, May 2012

Chapter 2

Ethernet in Embedded Systems: the Automotive Case

Embedded systems have evolved greatly over the last years. One good example is that of automotive systems, which we will take as inspiration. Automotive technology witnessed a shift from mechanical components to electronics to meet the demands for increased safety and customer comfort, e.g., to accomplish functions such as engine control, auto parking, safety in cases of over/under steering and comfort features such as air-conditioning. According to the work in [45] electronics in a car exceeded 20% of the total vehicle value in 2011 and reporting statistics from [46], the article [47] predicts the cost to be at 50% until 2030. Initially, car functions were implemented through stand-alone Electronic Control Units (ECU). However, to extend the functionality and leverage the benefits of data sharing, coordination was provided between ECUs with early solutions being point-to-point links. Using this approach, however, for a system with n ECUs, $n \times (n-1)$ cables were needed. This resulted in heavy cabling, increased weight, and an increase in the overall system cost. This problem was addressed by providing data exchange with a broadcast communication model such as the one provided by a serial communication bus [48, 49].

2.1 Challenges of System Integration within an Automobile

The electronics in a car are typically divided into *domains*, where each domain implements certain functionality [32]. Frequently, subsystems within the car are developed in a distributed fashion with different teams responsible for different system parts. In the automotive supply chain, we can identify the following important development roles. Original Equipment Manufacturers (OEMs), such as GM or Toyota, are responsible for the final product in the market. OEMs provide the product design specifications and finally integrate different ECUs to a functioning system. Tier-1 suppliers, such as Bosch or Siemens, provide subsystems such as power-train management or brake-by-wire subsystem to OEMs, and tier-2 suppliers are the chip manufacturers, such as Infineon or ST [50, 51].

This brings us to the discussion of challenges and opportunities which lie with the so-called *federated* or *integrated* architectures. In a *federated* architecture, each subsystem is implemented on its own stand-alone distributed hardware base, consisting of ECUs and communication channels (e.g., CAN). This would mean that each subsystem is independent, implementing its defined functions and communicating with the rest of the system through gateways. This leads to expensive solutions with many ECUs in the architecture. On the other hand, within an *integrated* architecture, the number of ECUs, sensors or wiring points can be reduced thereby reducing the overall system cost. For example, there are some sensors whose data are useful or necessary to the computation being done in different subsystems, e.g., road wheel sensor can be required/ essential to Antilock Braking System (ABS), instrumentation, Electronic Stability Control (ESC) and possibly others. Such sensors can be shared by the ECUs. However, a mechanism must be provided to reduce the influence of one subsystem on another. In particular, within an *integrated* architecture, principles of *composability* must be supported; services provided by a component before integration must remain intact after the integration and supporting non-interfering interactions (i.e. communication system must meet the component's temporal requirements regardless of temporal behavior of other components). The work in [52] presents a Time-Triggered System-on-a-Chip (TTSoC) architecture that helps integrating applications with different levels of criticality (non safety-critical applications with high criticality applications). Each subsystem is realized with one or more micro-components. A TTSoC may host several subsystems, *composability* is preserved since each micro-component is encapsulated; i.e., temporal and spatial interference is prevented by a special module Trusted Interface Subsystem (TISS) within each micro-component. TISS controls access to the Network-on-Chip (NoC).

Integration and isolation are particularly relevant for the introduction of x-by-wire systems due to their criticality. To address these concerns, redundancy in hardware and software components can be provided, increasing the overall cost. Alternatively, the work in [33] proposes employing verification and validation along the product design and development and supports component-based design and application analysis methodology for critical components. Another aspect is the existence of a large number of applications with diverse characteristics. Some control loops (e.g., break-by-wire, fuel injection) have stringent real-time requirements, while other applications require more computing power and a higher bandwidth (e.g., driver assistance systems). Finally, due to inter and intra-domain communications there is a continuous increase in the amount and heterogeneity of exchanged information. Today, such requirements are supported by using different networking technologies in various subsystems. For example, LIN with speed of up to 20 kbps, CAN up to 1 Mbps, FlexRay up to 10 Mbps, MOST up to 24 Mbps etc. An overview of automotive communication technologies can be found in [53]. The work in [50] emphasizes the need for methods and tools for system level analysis in order to verify the predictability or prove composability and hence support the designers that must evaluate and choose the system architecture.

2.2 Ethernet-based in-Car Communications

Switched Ethernet due to its wide-availability, high bandwidth and low cost is now being considered as in-vehicle networking technology, in particular, to provide a single shared backbone with increased bandwidth, integrating traffic from different domains. However, it must be able to support all the different requirements outlined above, in particular, regarding determinism in the timing behaviour. There are several approaches available now to achieve real-time communications over Ethernet in embedded systems including IEEE 802.1 AVB [22], FTT-SE [31], and TTEthernet [25]. The work in [54] lists similar communication challenges and studies in-car communication based on real-time Switched Ethernet backbone. The work in [55] studies the impact of low priority unshaped traffic (best-effort) on the timeliness of high priority real-time traffic when such traffic share communication links. This work investigates the performance of Ethernet standards AVB and TTEthernet, in particular, credit-based shaper of AVB and time-triggered traffic and rate-constrained shaping of TTEthernet. Evaluations are done via simulations in OMNeT++ [56]. The network consists of 7 switches and 15 ECUs (host nodes) with 100 Mbps links. Traffic includes real-time control traffic, bandwidth-intensive media streams, and cross traffic. Both protocols transmit the control traffic at the highest priority, i.e., for AVB with class A priority and TTEthernt with a TT schedule for the synchronous part or rate-constrained shaping for the asynchronous part. A dedicated ECU generates periodic bursts of cross traffic that share links with higher priority traffic. The maximum transmission unit (MTU) for the cross traffic is varied between 46 B to 1500 B. Time-triggered class gives the best (least) maximum latency and jitter for the control traffic in the presence of cross traffic owing to its inherent determinism and independence of concurrent traffic. AVB gives the lowest performance in this case. For media streams, however, the maximum values for both latency and jitter are smaller with AVB. Further, different strategies to improve latency of real-time traffic are proposed, such as limiting the size of MTU. The work in [57] addresses such a problem of selecting an MTU that maximises throughput in FTT-SE. Other suggestions include restraining the cross-traffic through shapers, using links with increased bandwidth, carefully selecting topology that reduces the number of hops on the route of critical messages, or finally allowing frame preemption to improve high priority latency as given with the Time-Sensitive Networking standard [58]. In an earlier work [59], authors considered a similar setup but without cross traffic; they used a model derived from real in-car network configuration and traffic traces. In that work, both technologies (AVB and TTEthernet) met the desired end-to-end latency requirement for control traffic, i.e. 100μs, whereas in [55] only the time triggered traffic could comply with such requirement in the presence of concurrent cross traffic load. Both works [59, 55] support the use of TTEthernet for transmission of critical control data.
The work in [32] proposed an architecture for inter-domain communications addressing the complexity of future Advanced Driver Assistance Systems (ADAS). The architecture partitions the applications into five domains; different domains connect with each other through a switched Ethernet-based backbone. Further, three topologies are presented for the network, and each architecture is evaluated using INET framework within OMNet++. Traffic includes real in-car traces of

CAN or FlexRay data as well as analytically modelled data. Not considering any particular real-time Ethernet technology, they evaluate two end-to-end latency requirements; a soft 10ms and a hard 100μs requirement. Two scenarios are considered, with or without cross traffic. Besides, prioritisation impact is examined. Soft requirements are met in all the settings, whereas for each topology, prioritisation improves the end-to-end latency. Hard latency requirements are violated with a small probability ($\sim$ 3%) even without cross traffic. An interesting result of this work is guaranteeing soft requirements in the presence of cross traffic utilising $\sim$ 80% link bandwidth.
To summarise the discussion, we refer to the work in [21] which presents an interesting survey motivating the case for Ethernet in automotive communications pointing to the two main aspects deemed inevitable in future automotive systems: the need for higher bandwidth, and a common network technology to reduce the complexity. Additionally, Ethernet supports the IP stack and opens the way to many exciting applications by integrating cars in the so-called IoT for example with a possibility to offload computation intensive elements to data centres outside the car.
In general, a car represents an interesting example of a complex distributed embedded system where multiple applications of mixed-criticality co-exist. Hence, different domains in the car may benefit from specific switched Ethernet-based technologies. In the remainder of this chapter, we present an overview of Ethernet, switched Ethernet and switched Ethernet-based technologies, namely, AFDX, AVB, TTEthernet and FTT-SE. This chapter concludes by giving a qualitative comparison of the different technologies and motivating our choice of using FTT-SE in our work.

2.3 Ethernet

Ethernet is a local area network technology developed in the 1970s whose first design principles can be found in [60]. Ethernet originally provided a shared medium (shared Ethernet) for the propagation of digital signals, and its construction was carried out using coaxial cables, twisted pairs or optical fibres (Figure 2.1). The stations were connected to a shared medium and arbitrated the network bandwidth in a distributed way. Ethernet offers synchronous serial communication. By synchronous, we mean that the clocks in the sender and the receiver stations are synchronised, and the data is sent at a constant rate. A transmitting station broadcasts bit sequences called packets onto the medium and expects that the intended receivers will hear them. A receiving station examines the destination address, and if the packet is not intended for itself, the station discards such packet.

2.3.1 Probabilistic nature of Ethernet transmissions

Ethernet was, initially, half-duplex; the medium was shared, and two or more stations could transmit packets onto the medium at the same time. Such packets would collide, interfering and becoming unrecoverable by the intended receivers. When the medium is idle, the transmissions are successful. However, as more stations begin to transmit, the interference and the packet losses increase. A single Ethernet segment is called a collision domain since all stations in that segment are connected to the same medium and prevented from transmitting during a collision resolution

process. After a collision, the sender aborts the current transmission attempt and tries to retransmit after a randomly chosen period. The randomly selected waiting period before retransmission reduces the probability of future collisions in case multiple transmissions are waiting to transmit.

2.3.2 Rationale for minimum Ethernet frame length

In shared Ethernet, all nodes in a collision domain must be able to detect collisions. In fact, if a station does not know that its transmission was involved in a collision, it may incorrectly decide that the frame was successfully sent and release the medium; hence the chance of a (possible) successful transmission is lost. When a sender detects a collision, it sends a special signal, called jam signal, to inform everyone that there was a collision.
The only time that an Ethernet controller can detect collisions on the wire is when it is in the transmit mode. When an Ethernet Network Interface Card (NIC) has finished transmitting and switches to receive mode, the only thing it listens for is the 64 bit preamble that signals the start of a data frame. The minimum frame size in Ethernet, typically 64B at data rates below 1Gbit/s, is specified such that, based on the speed of propagation of electrical signals in copper media, an Ethernet card is guaranteed to remain in transmit mode, and therefore detecting collisions, long enough for a collision to propagate back to it from the farthest point on the wire from it. Stations transmit when the channel is idle else wait; in case of collision, stations wait for a random amount of time and retransmit. This distributed arbitration mechanism is known as *Carrie Sense Multiple Access with Collision Detection* (CSMA/ CD).

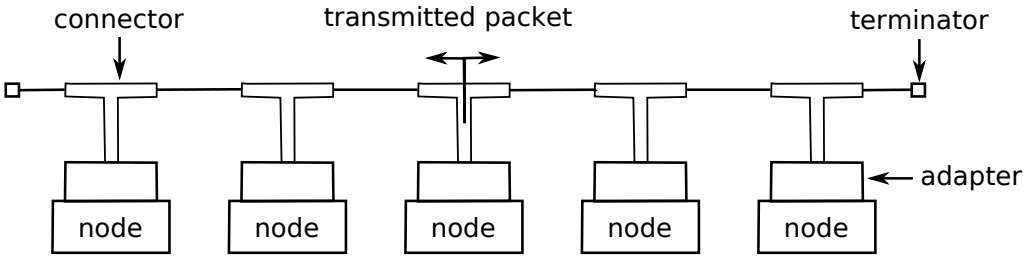


Figure 2.1: Half duplex Ethernet, nodes connected through a coaxial cable in a bus topology

2.4 Switched Ethernet

Switched Ethernet was introduced in the early 90s to overcome the problems faced by shared Ethernet, in particular, non-determinism inherent to CSMA/CD. Switches implement an independent collision domain per port; thus each port becomes a segment. When there is a single station connected to each port, we say the network is micro-segmented. Moreover, switches are now full duplex meaning that each link has two independent sub-links, one to communicate in each direction. Thus, the CSMA/CD arbitration is not needed[1], and stations and switch can transmit at the

[1] Modern Ethernet switches have ports that are full-duplex, however, CSMA / CD is still supported when legacy equipment is connected which operates in a half-duplex mode such as a hub, and where simultaneous transmission is treated as a collision.

same time without collisions, which allows many communications to co-occur, thus improving global throughput. Switches maintain an address table, and hence when an Ethernet frame arrives on one of its input ports, the switch can inspect the destination address, consult its address table to find the destination port, and forward data only to the appropriate port. If that port is busy, the message can be queued in memory and transmitted later (Figure 2.2). There can be multiple queues per output port to manage different classes of traffic. The queue scheduling policies can have a significant impact on the network timing behaviour [61].

2.4.1 Limitations of switched Ethernet for real-time communications

Traditionally Ethernet switches can handle message arrivals fast enough so that queues do not build up at the input ports. However, queues may build up at the output ports whenever several messages arrive in a short interval and are forwarded to the same destination port. Within traditional crossbar and memory designs, there is insufficient bandwidth to allow every input port to write into the same output queue simultaneously. Hence, such a situation may lead to overloads in which the output queues use up all the available memory, and further messages may be discarded. Another condition known as *head-of-line blocking* can occur with switched architectures. Sometimes the switches need to do deep packet inspection or shaping in the input ports, which takes time and slows them in handling the incoming packets. Thus, the incoming packets need to be held in input queues, before they are effectively forwarded to the output ports. With FIFO input buffers, the packet at the head of the queue is forwarded first. If its destination output port is busy, it cannot be forwarded. It is possible that packets back in the queue have different destinations and also on such destinations the output ports are not full. Thus, the input queues introduce the blocking and degrade system performance.

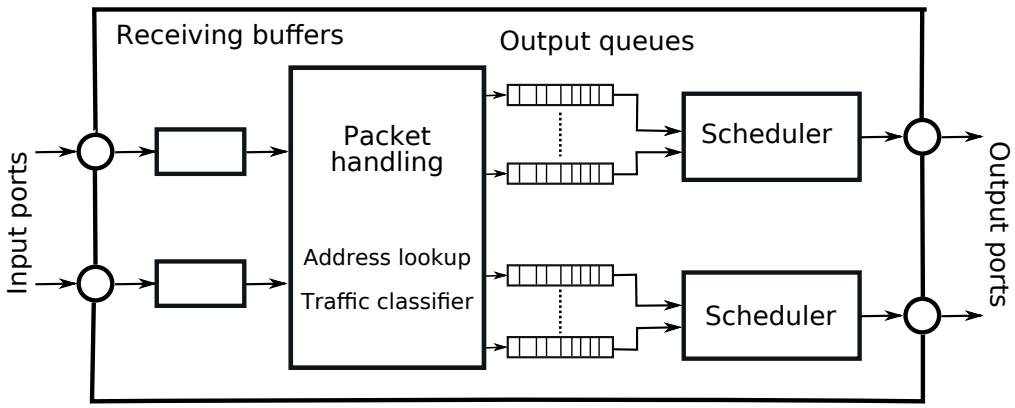


Figure 2.2: Typical switch internal architecture (Source: [31])

2.5 AFDX - Avionics Full Duplex Switched Ethernet

AFDX [62] is a data network standard used by aircraft manufacturers, such as Airbus and Boeing, to support safety-critical applications that need guaranteed bandwidth and deterministic quality

of service. Shared Ethernet was not suitable to meet such requirements due to the possibility of collisions and non-determinism inherent in CSMA / CD arbitration of the shared medium. Moreover, to meet the requirements for higher bandwidth resulting from an increase in the number of avionics applications, and the needs for cost savings, it is desirable to use commercial off-the-shelf (COTS) tools in Aircraft Data Networks (ADN), hence the use of switched Ethernet technology.
AFDX networks contain two types of devices, special Ethernet switches that perform traffic filtering and policing, and the end systems (ES) that send or receive data through the network. Since an aircraft is a closed system, AFDX statically defines the network topology and traffic flows. End systems exchange Ethernet frames using virtual links (VL). A VL represents a unidirectional connection from one source to one or more destination end systems. Logical isolation between different VLs is achieved by allocating a specific bandwidth to each VL. In this way, the available bandwidth to any VL is unaffected by the utilisation pattern of other VLs. A VL is characterised by its identifier, minimum period between two consecutive frames of that VL known as Bandwidth Allocation Gap (BAG) and the minimum and maximum lengths of VL frames. Different frames on the same VL can have different lengths comprised between min and max frame length specified.
To achieve deterministic behaviour traffic shaping is performed in the ES, i.e. limiting the data that each station may transmit guarantees that the switches are not overloaded whereas policing is performed in the switches. In particular, switches check whether the sum of the frequencies of all the messages transmitted on a given VL does not exceed the maximum frame frequency of the VL. From the ES perspective, there can be multiple applications sending data on the same VL. Each application generates frames at their rate which are scheduled according to Round-Robin scheduling and inserted into the VL FIFO queue that will be used to put frames onto the physical link (Figure 2.3). The number of applications per ES is limited to four, i.e., the number of sub-VLs that can be created per virtual link. Also, the use of round-robin scheduling handles applications without any sophisticated priority ordering. This is already an example of hierarchical scheduling, despite very limited. To prevent data loss, AFDX employs redundant channels in which the same data is sent on two networks simultaneously. Integrity checking on the end systems can remove duplicate or wrong data.

2.6 AVB - Audio Video Bridging Standard

Ethernet AVB is a set of extensions to the Ethernet standard that provides time synchronised, low latency streaming services for audio and video applications. AVB standard [22] achieves QoS requirements for these streams through a combination of stream reservation and traffic shaping mechanisms defined in the following IEEE standards.

- 802.1AS Timing and Synchronization for Time-Sensitive Applications (gPTP) [64].
- 802.1Qat Stream Reservation Protocol (SRP) [65].
- 802.1Qav Forwarding and Queuing Enhancements for Time-Sensitive Streams (FQTSS) [66].

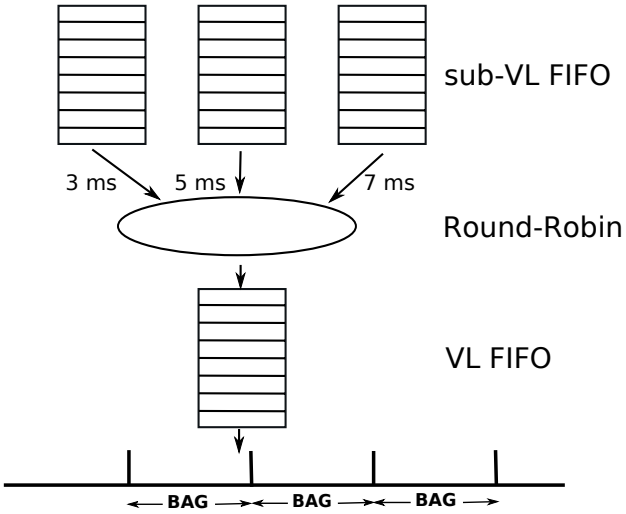


Figure 2.3: sub-VL scheduling in ADFX (adapted from [63])

- 802.1BA Audio Video Bridging (AVB) Systems [22].

2.6.1 An AVB system

An AVB network has the following devices: *talker*, a device which is the source of an AV media such as a microphone or a digital camera; *listener* is a device which receives the AV media from the network. Examples include speakers and monitor displays. A device can both be a listener and a talker. *endpoint* is the generic term for devices that produce or receive an AV media (talkers or listeners). *stream* refers to the flow of an AV media from a talker to a listener, and *bridge* refers to the switch that scales the network by connecting endpoints or other bridges.

2.6.2 Importance of synchronization

Precise timing and synchronisation are important for time-sensitive communication between distributed nodes. The work in [67] underpins the concept; a distributed algorithm to achieve synchronisation between events in a distributed environment is presented which relies on time-stamped message exchange between communicating processes and can achieve a total order of the events. Considering AVB, as the name suggests, target applications are the audio and video such as in-vehicle infotainment, large-scale AV installations like theatres, professional audio/video equipment etc. and more recent examples such as hybrid TV [68].
The synchronisation is pertinent within an AVB system primarily for the reasons of perceived value to the end user. Relative timing between picture and sound is important, and human senses are capable of detecting slight delays between audio and video [69]. An ITU recommendation stipulates the threshold for detectability between $+45$ms to -125ms [70] for traditional television. A positive value indicates sound is leading whereas a negative value indicates sound delayed. A recent work gives an overview of some media synchronisation standards [71] underlining its importance. Within an AVB system, synchronisation provides a common time base for sampling/receiving data

streams from a source device and presenting those streams at the destination device with the same relative timing. Imagine, a system that comprises a host node that is delivering data (the talker) and two nodes that comprise the left and right speakers (the listeners). When all three nodes are synchronised (share a single global clock), the left and the right speaker will produce sound synchronously.

2.6.2.1 gPTP protocol

The clock synchronisation protocol for AVB is standardized as IEEE 802.1AS (gPTP) [64] which is based on IEEE 1588 Precision Time Protocol [72]. The protocol selects one device in the network to act as clock master. The master clock establishes the reference time for the network. The protocol forms a *synchronisation spanning tree* connecting all the local clocks where the clock master referred to as *grand master* is the root of this tree. The synchronisation is achieved by precisely time-stamping packets as they leave the master and arrive at each slave node. PTP measures and compensates for any queuing or transmission delays. In particular, forwarding delays in the bridges and communication delays on communication links are measured. In this way, the protocol performs local clocks correction to bring these in agreement with the master clock, with an accuracy of less than 1 μs specified between seven or fewer time-aware systems.
We present an example of calculating communication delay on the link between any two directly connected time-aware nodes in the system. We consider, two bridges, *A* and *C* in an AVB network; bridge *A* is the *grand master*. For this purpose, three messages are exchanged between *A* and *C* (Figure 2.4).

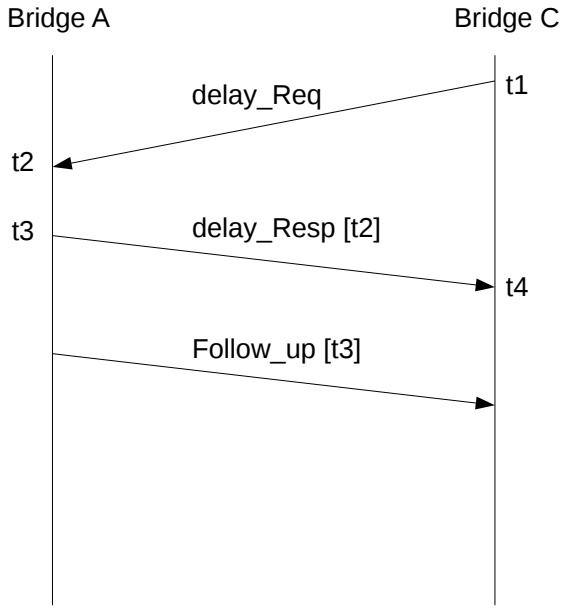


Figure 2.4: Synchronization mechanism and delay calculation

- bridge C sends a delay_Req message to bridge A and locally stores the transmission point in time t_1.
- bridge A timestamps t_2 when it receives delay_Req.
- at t_3 bridge A sends a message delay_Resp which also contains t_2.
- bridge C records point in time t_4 when it receives delay_Resp.
- Finally, to inform bridge C of t_3, bridge A sends a Follow_up message containing t_3.

After this exchange, bridge C has learned t_1, t_2, t_3 and t_4. Assuming symmetrical links, bridge C can approximate the communication link delay to bridge A as: $\frac{(t_4-t_1)-(t_3-t_2)}{2}$.

2.6.3 Stream reservation protocol (SRP)

SRP allows endpoints to reserve bandwidth across a compliant network [65]. Within a network that comprises AVB capable nodes (endpoints or switches), and legacy nodes, traffic shaping (streaming QoS) is provided only within the AVB cloud. However, devices outside the AVB cloud can communicate with the rest using standard Ethernet frames with best-effort QoS. SRP is based on Multiple Registration Protocol (MRP) defined in IEEE 802.1Q. This protocol enables streams to register attributes and distribute these across the network.
A network node (endpoint or bridge) that needs to communicate some data (i.e., a stream and its attributes) shall declare the information on all the connected network ports. The recipient nodes register the attribute information as well and declare it again. Each recipient repeats this process, effectively propagating the information throughout the network. Using Spanning Tree Protocol, MRP avoids loops and thereby congestion in the network. Figure 2.5 shows the principle of attribute propagation. In this example, the bridges are represented with squares labelled V through Z whereas circles (numbered 1 - 8) represent the endpoints. Node 3 is the source of attribute declaration. We can see that the attribute is registered and declared across the network.
In particular, a *talker* registers a *talker advertise* attribute that contains important stream information such as: StreamID, DataFrameParameters, TSpec, PriorityAndRank, and AccumulatedLatency.

Stream bandwidth requirement AV traffic can utilize upto 75% on each switch port. The bandwidth requirement for a stream can be calculated using TSpec which defines *MaxFrameSize* which is the maximum frame size in the stream, and *MaxIntervalFrames* which is the maximum number of frames within a *class measurement interval* (CMI). CMI can be regarded as the period of the class. The value of *class measurement interval* is 125μs and 250μs respectively for class A and B of AVB. With this information, the bandwidth BW for a stream is calculated as shown in Equation 2.1.

$$BW = \frac{MaxIntervalFrames}{CMI} \times 8 \times (MaxFrameSize + overhead) \tag{2.1}$$

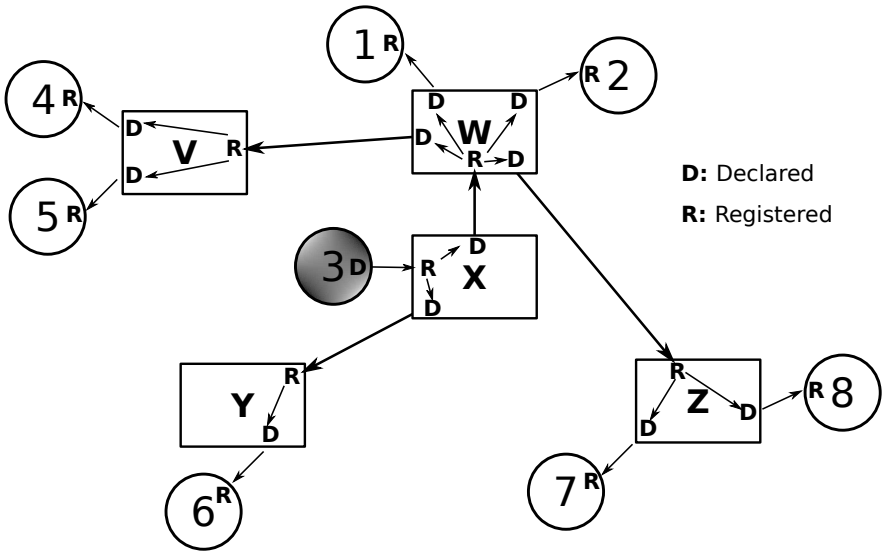


Figure 2.5: Attribute propagation through the network (adapted from [73])

The overhead of a streaming frame is 42 bytes and comprises Ethernet header, preamble, CRC and Inter-frame gap (Figure 2.6). Thus, for a stream with a frame size of 224 bytes, and considering $MaxIntervalFrames = 1$, the bandwidth requirement is about 17 Mbits/s in class A.

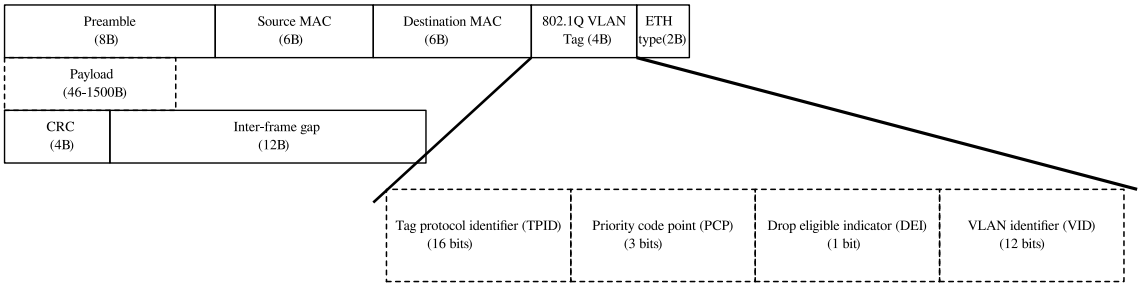


Figure 2.6: structure of an Ethernet frame detailing IEEE 802.1Q VLAN tag

Thus, a *talker* registering a particular stream involves the following steps: the *talker* declares that a stream is ready to transmit and registers *talker advertise*, which is propagated through the network as described. A *listener* that is interested in the stream shall register a *listener ready* attribute. This information is propagated across the network reaching the *talker* (Figure 2.7). A *bridge* receiving *listener ready* attribute on a port can check if enough bandwidth is available. This way, it is ensured that sufficient bandwidth is available along the path of the stream. Following the receipt of *listener ready* at the *talker*, streaming can commence.

2.6.4 Traffic scheduling in AVB

Ethernet AVB uses priority-based non-preemptive scheduling for messages. A set of messages with the same priority belongs to the same traffic class. Within a class, messages follow a FIFO

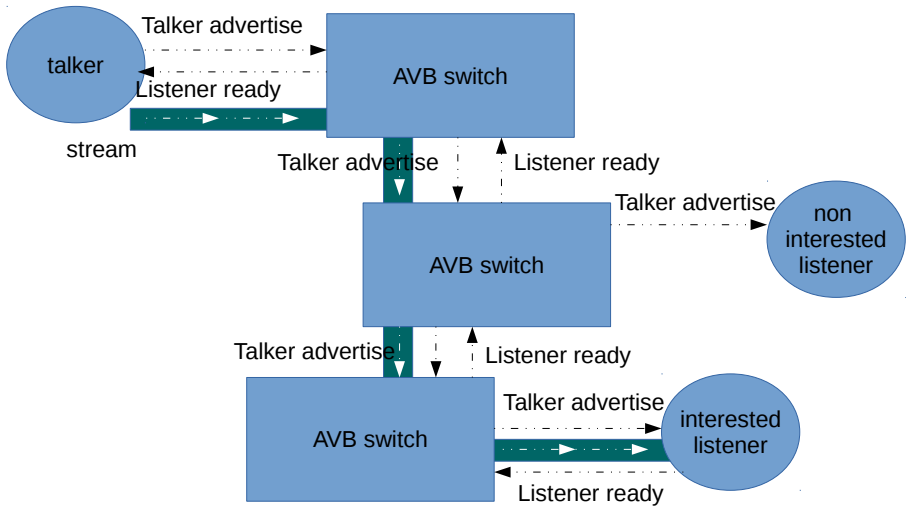


Figure 2.7: *talker advertise* and *listener ready* propagation (adapted from [74])

order. To prioritise different classes of traffic, AVB uses IEEE 802.1Q standard. A 3-bit priority code point (PCP) is used which refers to the IEEE 802.1p class of service (0 through 7) and maps to the frame priority level (Figure 2.6). Divided into three categories, traffic is scheduled either with a best-effort policy, with a credit-based shaping algorithm (CBSA) or with a time-aware shaping algorithm. Time-aware shaping is defined by the Time-Sensitive Networking (TSN) task group which we discuss in a later section. With these classes defined, the transmission selection function works as follows: A frame is selected from a queue q for transmission if:

- there is a frame available in q
- no queue with a higher priority traffic class has a ready frame

Above is true, however, for the best effort traffic only. For other classes, further conditions must be checked for message transmission.

2.6.4.1 Credit-based shaper

The credit-based shaping algorithm improves fairness between flows by avoiding bursts of AVB traffic, delaying messages according to some rules and thus low priority traffic can be served. CBSA applies to two stream reservation classes namely class "A" and class "B" corresponding to audio and video traffic respectively and with class A having a higher priority than class B. A frame of class A or B is selected for transmission only if the corresponding credit is non-negative. The rules for credit change are as follows:

- when no AVB frame (class A or B) is available, the corresponding credit is set to zero.
- when an AVB frame is transmitted, credit is decremented at the rate defined by *sendSlope*.
- when an AVB frame is waiting to be transmitted (either because another frame is in transmission or credit is negative), credit is accumulated at the rate given by *idleSlope*.

Figure 2.8 shows an example scenario of a bridge shaping traffic at an output port. In particular, it depicts three outgoing queues at a single output port. The queue for class A has three frames, whereas the queues for class B and other traffic (class C) each have a single frame. Frame C is on the link when frames A and B arrive. Since messages are transmitted non-preemptively, AVB frames have to wait, and they start accumulating credit at their respective rates. When, frame C completes transmission, two class A frames are transmitted next because class A has a higher priority than class B. We notice that another frame of class A becomes ready during the transmission of preceding class A frames. However, upon completion of preceding transmission, the credit has become negative so its transmission cannot begin and credit starts to increase. At this time, frame B is the highest priority frame eligible for transmission, and finally, the last frame of class A is transmitted. In this example, we can notice that delay in the transmission of frame A allows lower priority transmissions to commence.
The network latency requirement is 2ms for class A traffic and 50ms for class B traffic over seven hops (=six bridges) considering a 100 Mbit/s Ethernet.

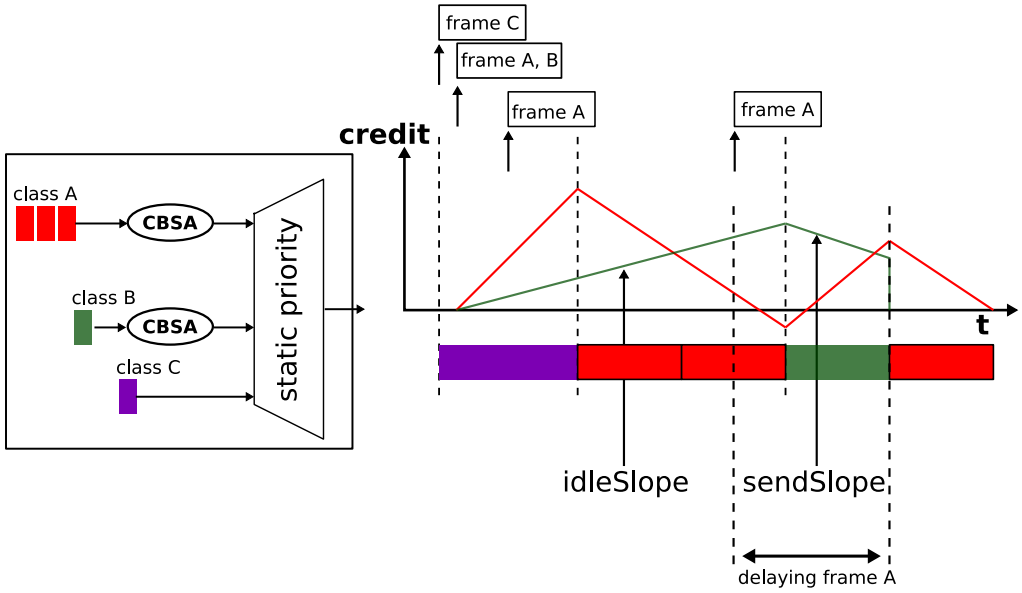


Figure 2.8: An example traffic shaping scenario at the output port of a bridge (adapted from [75])

2.6.5 Schedulability analysis for AVB

There has been substantial work regarding schedulability analysis of AVB. We refer to few such works here. The work in [76] presents an analysis for shaped traffic in AVB by considering so-called *eligible intervals* where there are pending frames and available credit for transmission, i.e., there is no idle time within an eligible interval. Two kinds of interference are considered, lower and higher priority interference subject to the impact of shaping and priority. Arrival patterns of the streams are not constrained. Non-preemptive transmissions result in lower priority interference in an AVB network. They analyse a given stream, separately, only with higher or lower priority interference, and compare the results to a base-line case when the stream is scheduled interference free.

The interference delays the stream by at most one interfering frame. This work does not consider simultaneous interferences (combination of higher and lower priority). An analysis considering the impact of mixed interference (from higher and lower class) on a medium priority class is discussed in [77]. Another work reported in [78] presents a busy period AVB traffic schedulability analysis. Factors influencing WCRT include interference by the traffic shaper (delay caused due to the rules governing credit recovery and consumption), FIFO interference from other messages in the same class, blocking by lower priority messages due to non-preemptive frame transmission, and interference by higher priority class. To build their analysis, they use so-called *phases*, where a phase is the time interval that begins and ends with a zero credit, or begins with a zero credit and ends with message (under analysis) transmission, known as the final phase. They formally show that lower priority may impact on shaped traffic (class A or class B) response time once, in the last phase. For class A, the higher priority interference does not exist. Traffic shaper impact is accounted by considering a minimum initial credit. Such credit is obtained considering a prior maximum size message transmission in the same class. Finally, all the delays are combined to derive equations for computing WCRT of shaped traffic in AVB.

2.7 Time Sensitive Networking (TSN)

Time Sensitive Networking (TSN) is a set of standards developed by the Time-Sensitive Networking Task group (IEEE 802.1). Its focus is to provide low latency and deterministic communications. In particular, the relevant standard is IEEE 802.1Qbv which supports a new type of traffic, so-called Scheduled Traffic [58]. TSN also improves the reliability of clock synchronisation mechanism thereby supporting safety critical and control applications, in principle, by introducing a redundant *grand master* in the network. With the presence of a primary and backup *grand master*, the standard provides methods for synchronised clocks between the two, hence, allowing a seamless transition from primary to the backup under failure conditions.

2.7.1 Time aware shaper

It is necessary to reduce interference from higher or lower priority traffic classes to reduce communication latency and communication jitter. Time Aware Shaper (TAS) helps realise this objective. TAS allows time-based forwarding of traffic. According to the schedule defined for time-aware traffic, the mechanism guarantees that at scheduled times, other queues get disconnected from the transmission selection function. Hence, the port is idle at those times to transmit frames of the time-aware traffic, thus, guaranteeing minimal latency. Figure 2.9 depicts the scenario at an output bridge port with two queues: a higher priority queue with three time-aware frames (TA1-TA3) and another low priority queue with frames (O1 and O2). The functionality of time-aware shaper is enabled with a TA signal; when enabled, time-aware frames are transmitted, else lower-priority traffic is transmitted. With time-aware shaper, benefits of cut-through switching can be gained too as the frame forwarding process can begin as soon as the destination is known without having to receive the complete frame first [79].

Figure 2.9 also shows a time-interval called *guardband* before the start of time-aware transmission window. The purpose of *guardband* is to avoid conflict between traffic agnostic to time-aware shaping and time-aware traffic. If a non time-aware frame starts transmission just before the reserved time window for critical traffic, its transmission might extend into the reserved window and cause critical transmissions to extend outside the window. Therefore, a *guardband* equal in size to the maximum sized frame is placed before the reserved window starts (i.e., TA is enabled) (Figure 2.10). Other frames can begin transmission if such transmission can be completed before TA is set to be enabled.

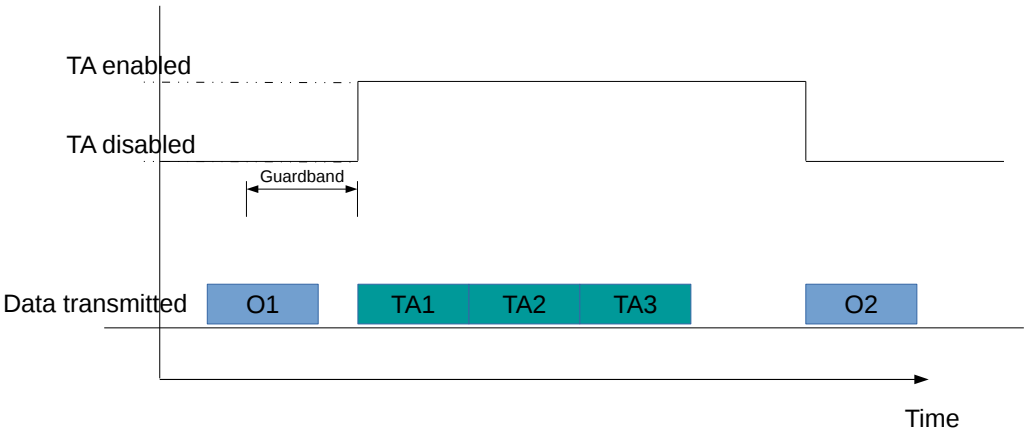


Figure 2.9: An example of time-aware shaper (Source: [73])

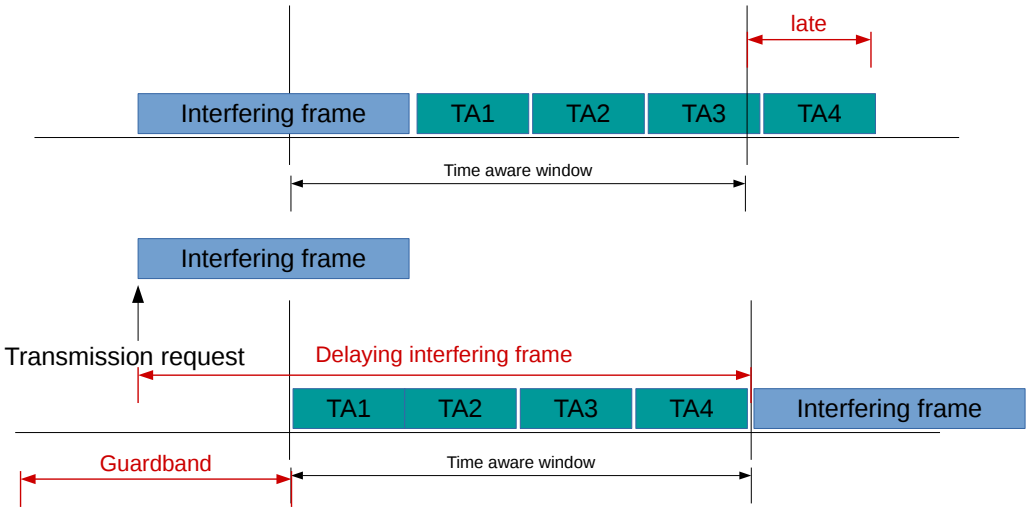


Figure 2.10: Time-aware shaper, guardband before critical transmissions (adapted from [80])

The *guardband* may, however, result in idle times and therefore bandwidth waste, for example, when transmission starting within the *guardband* cannot complete before TA is enabled. Such a situation delays the transmission and some idle time appears in the link until the start of the reserved window. There have been different approaches to make efficient use of the *guardband*.

One method that allows starting transmissions within *guarbband* would require shorter frames as we advance in time to the point where TA is enabled. Which implies searching outgoing queues to find eligible frames. Another approach uses preemption. With this approach *guardband* can be as large as the size of the largest fragment instead of the largest interfering frame. The preempted frame can resume transmission after the reserved window (Figure 2.11).

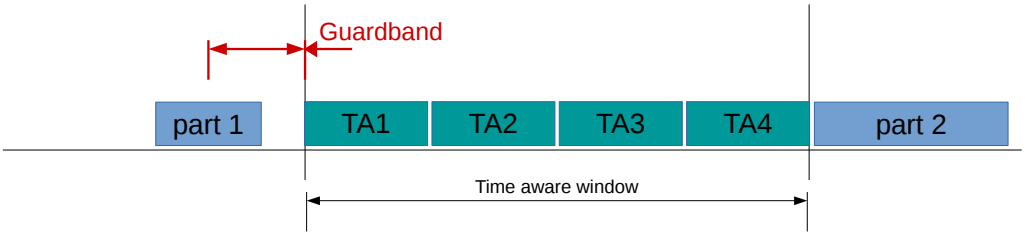


Figure 2.11: reducing the guardband, preemption approach (adapted from [80])

2.8 Time-Triggered Ethernet (TTEthernet)

TTEthernet [25] is a real-time Ethernet technology with the primary design objective to support applications with strict temporal guarantees over standard Ethernet. Further, it aims at providing a unified communication system supporting different traffic types (e.g., real-time, non-real-time, bandwidth intensive) as well as integrating time-triggered and event-triggered traffic while being compatible with standard Ethernet.
The protocol defines three traffic classes of different criticality namely Time-Triggered (TT), Rate-Constrained (RC), and Best-Effort (BE) traffic. Time-Triggered traffic has the highest priority and dispatches messages according to a fixed predefined schedule. Rate-Constrained transmission has a lower priority than TT and is executed when TT communication is not present. RC traffic provides a bounded end-to-end latency and specifies a minimum time interval between two consecutive instances of a frame on the same data flow. Best-Effort traffic has the lowest priority, being transmitted in a standard Ethernet communication paradigm with no delay constraints or temporal guarantees. The highest priority hard real-time communication is carried out using TDMA based bandwidth partitioning. Communication is organized in communication cycles known as *major time frame*.

2.8.1 Architecture model

TTEthernet network is a multihop switched Ethernet network organized in clusters, where a cluster comprises End Systems (ES) that are interconnected with links and Network Switches (NS). To explain, we refer to Figure 2.12 (borrowed from [81]) depicting an example cluster with four end stations $\{ES_1 \cdots ES_4\}$ and two switches $\{NS_1, NS_2\}$. The cluster can be modeled as an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{ES_1, \cdots, ES_4, NS_1, NS_2\}$ is the set of nodes in the system and $\mathcal{E}$ is

the set of full duplex connections. A *dataflow link* $l_i = [v_j, v_k]$ is a directed communication connection between any two adjacent nodes. A *dataflow path* dp_i is an ordered sequence of dataflow links connecting a sender and a receiver. For example, dp_1 in Figure 2.12 can be denoted as $[l_1, l_2, l_3] = [[ES_1, NS_1], [NS_1, NS_2], [NS_2, ES_3]]$. Further, the cluster has two applications App_1 and App_2; App_1 is a highly critical application with task set $\{\tau_1, \tau_2, \tau_3\}$ whereas App_2 is a non-critical applications with task set $\{\tau_4, \tau_5\}$. Tasks in the task set are mapped on different ESes. Within the cluster, physical links are shared by tasks of different criticality, e.g., messages from τ_1 and τ_4 can interfere in the links l_2 and l_3. Critical communication that takes place within $\{\tau_1, \tau_2, \tau_3\}$ needs to be protected from non-critical communication. This is made possible by using *virtual links* which are logical unidirectional communication channels from one source to one or more destinations as defined in [62] i.e., vl_1 and vl_2 isolate critical from non-critical traffic.

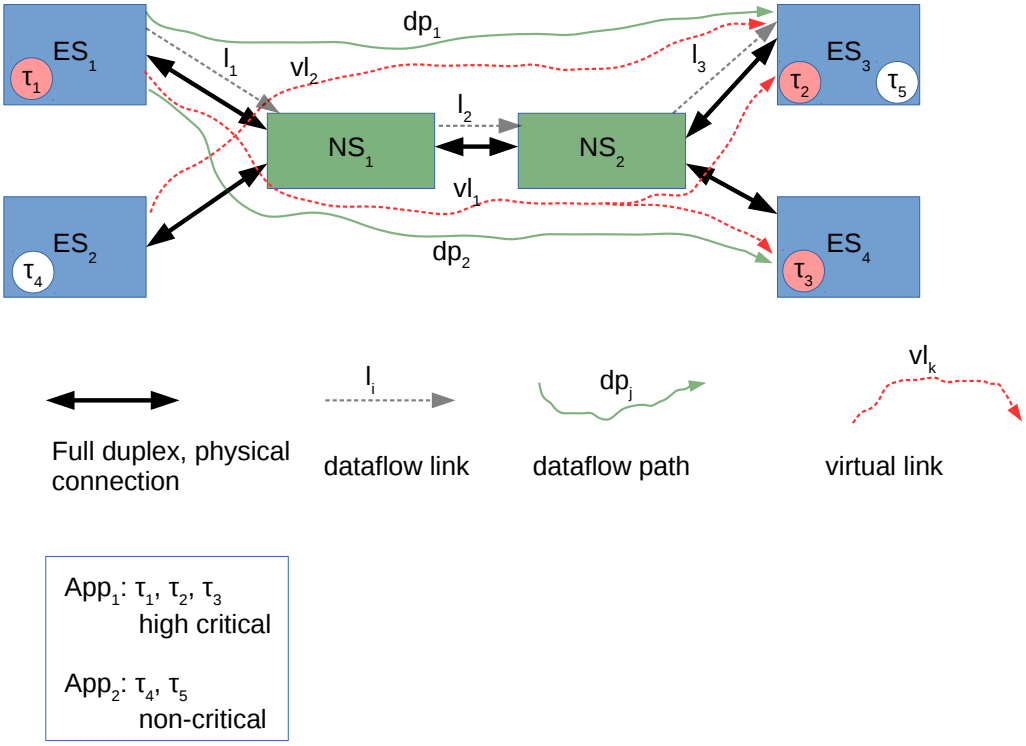


Figure 2.12: TTEthernet cluster example (Source: [81])

2.8.2 Traffic scheduling

In this section, we explain, the scheduling of TT and RC traffic. Precise clock synchronisation is a pre-requisite for building TT traffic scheudles [82]. A complete set of schedules within a cluster is denoted by $\mathscr{S}$ which contain the sending times and receiving windows for all frames transmitted during a *major time frame*. The end systems maintain a sending schedule table S_S whereas the network switches maintain both a sending table S_S and a receiving table S_R. Communication schedule is built and distributed offline; scheduling tables are stored within each ES, and NS of the cluster.

The schedule ensures that transmissions from different nodes on shared links are separated in time to avoid conflicts and congestion. Building TT traffic schedules, however, is a complex problem, for several reasons such as large network, the presence of a large number of TT messages, or dependencies between specific messages.

When an ES has a time-triggered message to send, its local scheduler consults the scheduling table S_S, and forwards the frame to the next hop at the configured time. When the frame arrives in an NS, the filtering unit within the switch identifies its traffic class based on a bit pattern specified in the frame header. NS now consults its scheduling table S_R to check if the frame arrived within the valid time window, and if it is a TT frame, then it is passed on to a TT scheduler to be forwarded to the next hop at the configured time. If the received frame is outside the window, or if it is a duplicate frame, then it is dropped, thereby implementing containment/fault tolerance within the NS. It is important to note that the duration of the time interval that the switch allows must account for a synchronisation error between the source and the switch.

In contrast, RC transmissions are event-triggered, and no scheduling tables are maintained for this class. The separation between different RC messages is achieved with bandwidth allocation. For a virtual link vl_i that carries an RC frame f_i, the designer chooses a minimum time interval between successive instances of f_i. This interval is known as *Bandwidth Allocation Gap* (BAG). The source ESs enforce the BAG. Thus, an ES will ensure that for each BAG_i, there is at most one instance of f_i. Thus, even when a task within an ES generates bursts of frames, these shall leave the ES within specified BAG. Since RC messages do not follow a deterministic schedule, different sources may create RC messages to the same destination simultaneously. Such messages will queue up in network switches resulting in increased jitter (Figure 2.13). Additionally, NS also implement a *leaky bucket* algorithm that ensures that the time interval between successive instances of a frame on a vl_i, compensating for the maximum allowed jitter, is not shorter than the BAG time. The algorithm drops the frame that violates such condition.

Finally, we consider the scenario where the RC and TT traffic is mixed. In principle, the protocol transmits RC traffic only when no TT traffic is present. However, as noted earlier, an RC frame can be sent almost at any time, and thus interference can happen at the link access. Such a situation may arise when a TT frame is scheduled to be transmitted, but an RC frame is already transmitting. To handle such cases, different integration policies are defined namely *shuffling*, *preemption*, *timely block* and *resume preemption*. With the *shuffling* approach, when a TT frame arrives, an ongoing RC transmission is not preempted. Instead, the RC frame completes its transmission, and later the TT frame is transmitted. The implication is that now the TT frame transmission is no more associated with a particular time instant, rather to a scheduling window. With the *preemption* approach, also known as *preemption restart*, RC transmission is aborted, and the arriving TT frame is transmitted instead. After TT frame finishes transmission, the complete RC frame is resent. With *timely block* approach an RC frame is blocked from transmission if a TT frame is scheduled during the transmission of this RC frame. With *resume preemption* a preempted RC frame resumes its transmission from the point where it was stopped. A work in [84] studies the impact of different integration policies on the delay performance of RC and TT traffic.

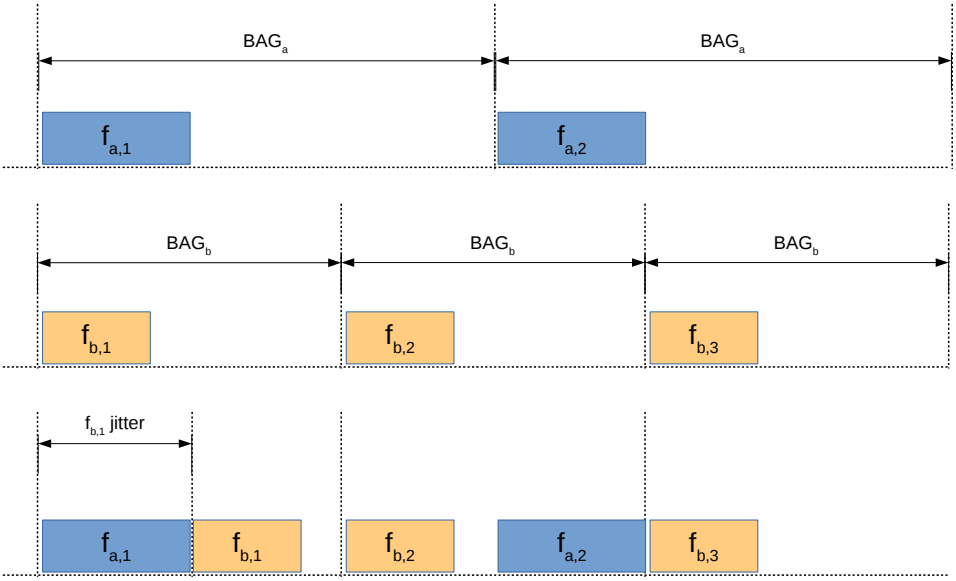


Figure 2.13: TTEthernet rate- constrained (RC) traffic (Source: [83])

2.9 FTT-SE: a Brief Overview

FTT-SE [31] is a master/slave protocol for real-time communication on Ethernet that exploits the advantages brought by micro-segmentation on typical star-topologies, namely parallel forwarding paths and absence of collisions. The protocol organises communication in fixed duration slots called Elementary Cycles (ECs). The EC duration is a design-time parameter, tunable to balance the application reactivity requirements and the associated overhead. Typical values range from 1 ms to tens of ms. Each EC starts with a Trigger Message (TM), issued by the master, which contains the schedule for that interval. The remaining nodes in the system receive the TM, decode it and transmit the messages indicated therein. Each EC is further divided into two windows, for synchronous and asynchronous traffic classes, respectively (Figure 2.14). The share reserved for the synchronous traffic is also a design-time fixed parameter. Typically, the master first schedules the synchronous traffic up to the synchronous window and only then schedules asynchronous one, using the remaining time in the EC. Windows overruns are not allowed by schedule construction.

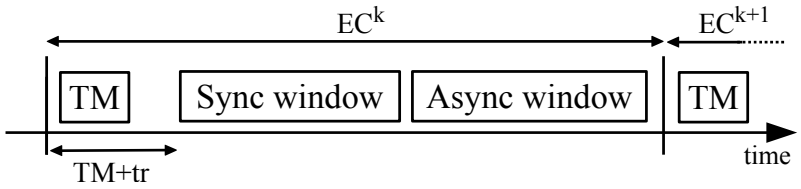


Figure 2.14: The FTT-SE EC structure

2.9.1 Handling synchronous & asynchronous traffic

The synchronous (or isochronous) traffic is periodically activated within the master node that schedules according to some policy, e.g. using EDF or RMS. The Master naturally enforces the traffic properties, being the slaves only task responding to the TM transmission orders. As opposed to the synchronous one, which is triggered autonomously by the master in an isochronous way, the asynchronous traffic is triggered by the slaves in an event-driven fashion. Therefore, FTT-SE provides a signalling mechanism that aggregates transmission requests during each EC and conveys them to the master in a specific minimum sized packet (see [85] for details). Once the asynchronous requests arrive at the master, from all nodes in parallel, they can be scheduled with any desired policy. The extra latency implied by this signalling mechanism ranges from 1 to 2 ECs.

2.9.2 Building traffic schedules

The master centralises the scheduling activity, allowing a smooth deployment of any scheduling policy as well as performing atomic changes to the communication requirements. This last feature facilitates the implementation of mechanisms that enable admitting and removing message streams, online, under guaranteed timeliness, and mechanisms for dynamic bandwidth management.
When scheduling, the master uses a special transmission time accountancy per link to make sure that the traffic indicated in each TM can be fully transmitted within the respective EC (see [86] for details). This feature makes sure that all switch queues are empty by the end of each EC and the traffic pattern at an EC scale follows the scheduling performed by the master. This is achieved by using two virtual bins per port. The capacity of each bin is equal to the length of the respective time window. One bin accounts for the *uplink* load while the other accounts for the *downlink* load. While preparing a schedule, the system scheduler picks one message at a time from the ready queue, computes its size and checks if it fits within the respective bins in the uplink and the downlink. In detail, uplinks are tested simply by using the accumulated load from the respective source; in the downlink, the scheduler takes into account the offset from the source and the impact of jitter from interfering messages in the downlink. If the latest finishing instant for the message under consideration accounting for all such delays is within the window and already scheduled load in that bin is not affected, then the message is added to the schedule (Figure 2.15). Messages that fit are scheduled and encoded in the TM which is broadcast by the master at the beginning of each EC.

2.10 A Qualitative Comparison of Different Technologies

This section presents a brief qualitative comparison of the studied technologies, i.e., AFDX, AVB, TTEthernet, TSN and FTT-SE. Looking at some aspects that are relevant in the design of efficient CPS, we study how each technology fares in these aspects.

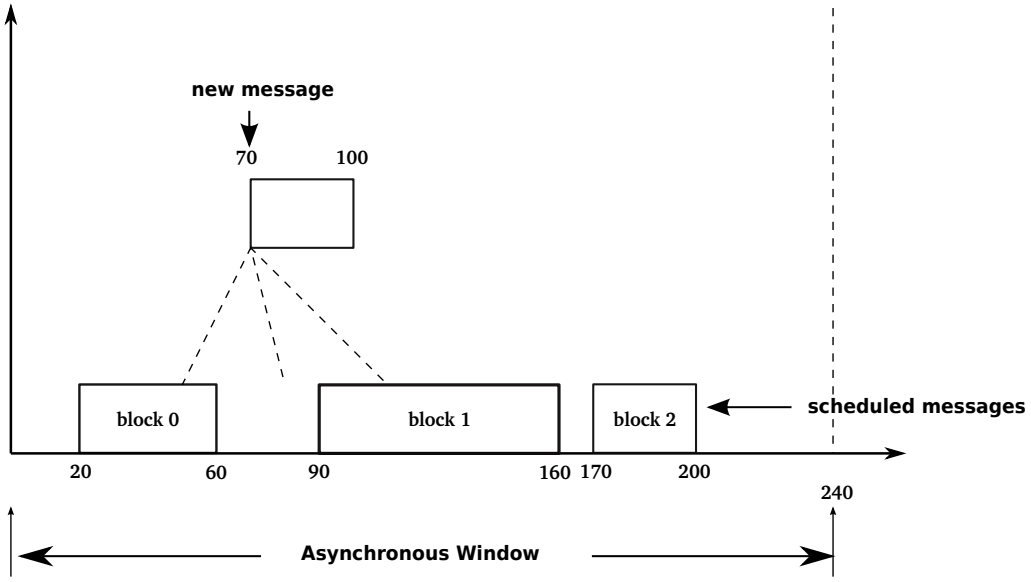


Figure 2.15: Adding a message to a downlink, checking against the end of asynchronous window

- **Supported priority levels**
  When multiple messages compete for the link access, priorities allow the conflict resolution in the switch output ports. We can find two situations where switch needs to resolve the conflict:

  1. Two messages are received simultaneously in different input ports and need to be routed through the same output port, or several messages are waiting to be transmitted in the output port. Priorities determine how these messages shall be serialized.
  2. A message transmission is ongoing, and another message is ready to be transmitted through the same output port. If the protocol allows preemption, then a higher priority packet can interrupt a lower priority transmission.

  This, in turn, impacts on the performance that different applications receive as well as the number of applications with different criticality levels that can be accommodated.

  – AFDX virtual links have one of the two priority levels, high or low. AFDX switches implement a FIFO scheduling at the output ports. This limits admission of traffic with multiple criticality levels. However, AFDX use in the safety-critical avionics domain requires an offline definition of the flow characteristics and a static schedulability analysis to ascertain the system timing. Thus, the system behaviour is guaranteed to remain deterministic at runtime. The work in [87] presents a QoS aware AFDX that combines fixed priority and FIFO scheduling in the switch output ports. This work analyses the impact of additional lower critical load on the performance guarantees of

critical avionic flows using only the FIFO or a combined FP/FIFO policy. Such impact is smaller with the combined scheduling. Priority based scheduling can improve worst-case response times and minimise needed buffer size in switches [88].

- AVB uses PCP, a 3 bit field within IEEE 802.1Q VLAN tag, which defines up to eight different priority levels. It determines the traffic class (=queue in which to place the frame). Audio and video streams receive a higher priority; other traffic can use remaining classes. When AVB is extended to support highly time-sensitive control traffic (or the scheduled traffic (ST) as in TSN standard), scheduled traffic is assigned the highest priority. Mapping critical flows in the ST class has clear latency improvement in comparison to when mapping these in AVB class A [89]. When frame priority is mapped into classes, multiple frames from different flows mapped in the same class suffer from mutual interference. Highly sensitive time-critical flows mapped to AVB class A can suffer unacceptable delays; being delayed by shaper or by large class A frames.
- Within TTEthernet, priorities are meaningful when scheduling rate-constrained traffic, as the time-triggered traffic follows a defined schedule. The standard allows multiple priorities for the rate-constrained traffic. However, most works consider three priorities, i.e., high, medium and low, each assigned to a traffic class [90, 91]. Contentions typically occur in the switch outgoing ports, which are resolved by serving messages belonging to the same class in FIFO order. TTEthernet employs specific integration policies to handle the scenario when an RC frame transmission is ongoing, and a TT frame arrives (refer section 2.8.2).
- FTT-SE assigns the highest priority to the time-triggered synchronous traffic. The unconstrained aperiodic communications (asynchronous traffic) can either be real-time or non real-time. The medium priority is assigned for real-time asynchronous traffic whereas non real-time traffic is assigned the lowest priority. Within an EC, a time window is reserved for the synchronous traffic. Asynchronous traffic can be transmitted in the remaining time in respective priority order. Since traffic scheduling activity is centralised to the master node, it is possible to enforce any scheduling policy for the streams of any traffic class.

- **Supported traffic types**

  - AFDX supports event-triggered traffic only, but with timing guarantees.
  - AVB supports two event-triggered traffic classes namely SR class A, and SR class B, using stream reservations. These classes are given bandwidth guarantees with defined limits on the delay. AVB also supports event-triggered best-effort (BE) traffic without temporal guarantees and time-triggered (scheduled) traffic with the TSN standard. With the support for scheduled traffic, priorities are overruled by the schedule. Achieving entirely deterministic communications amounts to creating offline schedules that guarantee low jitter and deterministic latencies for critical traffic [92].

  - TTEthernet supports three traffic classes, time-triggered (TT) traffic that follows a predefined schedule, rate-constrained (RC) traffic that is event-triggered with a defined bandwidth guarantee and bounds on delay, and best-effort (BE) traffic with no delay guarantees.
  - FTT-SE supports both time-triggered (synchronous) and event-triggered (asynchronous) traffic classes. It also supports non real-time traffic which is the asynchronous traffic without real-time requirements.

- **Bandwidth reservation**
  Communication protocols need QoS mechanisms beyond simple priority assignments to meet the diverse communication requirements. The goal is to manage the interference on shared links, thus providing temporal isolation among different applications. Different techniques allow achieving this objective such as either reserving strictly periodic windows as in time-triggered traffic or controlling the rate at which traffic is submitted in the network, for instance, shaping by CBSA in AVB or limiting the bandwidth through virtual links.

  - Virtual links within AFDX specify the maximum available bandwidth to the applications. For any virtual link, two parameters namely BAG and the maximum admissible frame size on that VL, determine the available bandwidth (bandwidth = max frame size / BAG). The BAG limits the bandwidth only, but there may be BAGs without any message to transmit[2]. The standard defines BAG values in milliseconds from the set $\{1,2,4,8,16,32,64,128\}$. When multiple applications send data through the same VL, designers must carefully choose BAG value to meet worst case scenario (multiple messages arriving within the same scheduling period). Also, the combined bandwidth of all VLs should not exceed the link rate, e.g., 100 Mbps.
  - Within AVB, bandwidth reservations are provided using the stream reservation protocol (SRP) [65] for streaming classes A and B. At the most, 75% of the total bandwidth can be reserved.
  - Within TTEthernet, time-triggered traffic schedule is calculated and distributed offline. Precise time synchronisation is necessary to route messages respecting this schedule. Thus, nodes obtain exclusive network access at specific time periods (slots) leading to highly deterministic systems. Bandwidth is reserved for the Rate Constrained (RC) traffic class, and its management is similar to the AFDX protocol, i.e., using logical point-to-point connections (virtual links vl). BAG is enforced by the ES i.e.; even when the task may generate a frame f_i in bursts, each BAG interval must contain at most one instance of f_i, thus respecting the bandwidth limit (size of f_i / BAG). However, RC traffic has a lower priority than the Time-Triggered traffic and may suffer delays in

[2]This situation may occur, for instance, when there are multiple messages multiplexed on the same VL and frequency of VL scheduling (its period) is selected to accommodate the worst case scenario, i.e., all messages arriving simultaneously. With such configuration, however, not all scheduling instants of the VL will find messages to transmit.

cases when a time-triggered message is transmitted simultaneously through the same port.

– FTT-SE allows configuring separate reservations for the synchronous and the asynchronous traffic classes. The share for each reservation is a design-time parameter, configured according to the application requirements. Further, individual messages can be allocated reservations within the share for the asynchronous traffic class.

- **Synchronization**

  – AFDX end systems do not share a notion of synchronised local clocks or a common global time reference.

  – The clock synchronization protocol for AVB is standardized as IEEE 802.1AS (gPTP) [64]. It provides a synchronisation error of less than 1 μs between seven or fewer time-aware systems.

  – TTEthernet needs precise clock synchronisation for building communication schedules for the highest priority time-triggered traffic. For this reason, it uses a failsafe fault-tolerant synchronisation protocol SAE AS6802 [93]. Such schedules are prepared and distributed offline and assign dedicated timeslots to each participant in a TDMA fashion.

  – FTT-SE relies on synchronisation achieved through the Trigger Message (TM). Master broadcasts TM at the beginning of the EC, which contains the schedule for that EC. Since all queues are empty by the end of previous EC, all slaves can receive TM without being impacted by the queueing delays. Further, *guard window* and *turn-around time* temporally isolate TM from the application traffic. Master does not specify any offsets for slaves transmissions relative to the transmission of TM. The schedule building takes into account that all indicated messages will finish transmission within that EC. If any message transmissions in the EC overlap, these messages are serialised in the switch queues.

- **Fault-tolerance**

  – Within AFDX, redundant channels are used. For each VL, identical frames are sent on two independent networks. Thus, under normal operation, an ES will receive two copies of each frame. The replicas are identified using a 1 byte *sequence number* field associated with the virtual link. The receiving ES performs an integrity check to see that the frames received from a network are in order. Following this, it performs the redundancy management which decides whether the frame should be accepted or dropped because its replica has already been received.

  – Currently, no redundancy has been defined for AVB. However, to be a suitable technology for the mission-critical systems, TSN standard must support fault-tolerance apart from the determinism. The standard can combine SRP with redundancy protocols to

realise fault tolerance. Stream reconfiguration time, as well as the time needed by the redundancy protocol, must be bounded and pre-determined. To this end, the work in [94] explores two approaches, one that integrates mechanism of the redundancy protocol inside SRP, and the other that decouples such mechanisms.

- TTEthernet provides fault-tolerance service at different levels. Scheduler task within an ES isolates flows of different criticality and transmits frames by strictly following the local schedule table even when the corresponding task may generate more frames than scheduled, for instance, by becoming faulty. Likewise, the task within the NS drops any frames arriving outside the designated window. Within TTEthernet, a virtual link can duplicate messages through multiple redundant switches for increased fault-tolerance.
- FTT-SE, currently, does not provide fault-tolerant services to recover lost messages such as TM, or the slaves request messages. Also, the switch, the Master and the Master communication link represent single points of failure in FTT-SE. Fault-tolerance aspects of the FTT-SE architecture were studied in the work [95], under the projects [96, 97], in particular, for the HaRTES architecture [98, 99].

- **Dynamic QoS**
  In the following, we see which technologies that provide the flexibility of changing the QoS of the communications online.

  - Dynamic reconfiguration is not supported by AFDX; the flows are defined statically considering the fixed worst-case requirements and thus remain unchanged during the system lifetime.
  - AVB supports reconfiguration of established streams. However, to update the stream attributes, the talker must tear down the stream first, wait a specified amount of time that is configured in the protocol, and then request the creation of this stream with new attributes [100]. During this waiting time, other streams can be registered by different nodes and may exhaust the available resources, thus potentially preventing the creation of the stream with updated attributes. Moreover, the protocol does not support reconfiguration of a given stream to a different traffic class.
  - TTEthernet does not consider TT traffic to be flexible. Changes in the TT schedule require global changes that are hard to enforce. However, if the application does not use the bandwidth allocated for synchronous TT messages, it is dynamically released and can be used for the asynchronous network traffic (RC or BE). For RC traffic, bandwidth is defined through virtual links with offline configured BAG and maximum frame size transmitted through the VL. The bandwidth usage respecting this limit is enforced at runtime. This configuration remains fixed through system runtime. Thus, TTEthernet does not support online changes to the communication characteristics of RC flows.

– FTT-SE allows reconfiguring the QoS dynamically. Slaves can send a message to the master requesting a change in the communication requirements, i.e., updating the stream period or the size of the transmitted message. Master evaluates if sufficient resources are available to carry out the change, in all the links. If it is possible to make the transition, then, new parameters are broadcast to the slaves to update their local databases. However, changing the traffic class of the stream, i.e., changing from synchronous to asynchronous, is not supported.

Figure 2.16 shows a relative position of each technology on the spectrum of communication determinism and QoS.

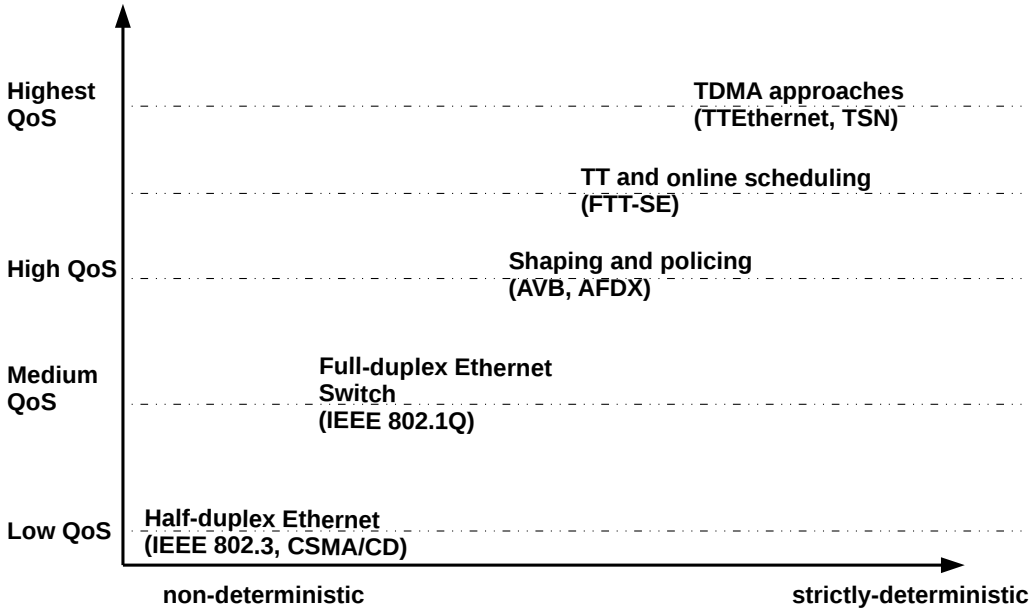


Figure 2.16: Determininsm and QoS in Ethernet-based technologies (adapted from [101])

2.10.1 Further remarks

We have studied different technologies that offer real-time communications over switched Ethernet, namely AFDX, AVB/TSN, TTEthernet and FTT-SE. In general, these technologies handle real-time applications using one or the other paradigm of communications, i.e., Time-Triggered (TT) approach or the Event-Triggered (ET) approach. Each approach has its merits and limitations. Using the TT approach, nodes obtain network access at specified time periods (slots). A TT approach offers several benefits such as:

- minimising the buffers inside switches as the schedule is free of conflicts,
- offering temporal partitioning and a composable solution since participants get exclusive network access at designated times.

Thus, a TT approach leads to highly deterministic systems and is ideal for systems with stringent timing requirements. However,

- performance of the time-triggered approach relies heavily on precise clock synchronisation services. A fault-tolerant clock service is needed.

- channel bandwidth is available on a periodic basis and is exclusive. When respective traffic is not pending, other traffic cannot use the bandwidth, thus, the bandwidth is wasted.

- building global communication schedules is a complex problem which involves using non-trivial optimisation algorithms, and which can be verified on minimal systems only [83, 84]. Modern systems will naturally consist of several applications and many messages, which potentially limits the scalability of a TT approach.

In contrast, with an ET approach, communication is initiated in response to certain events which may arise at any time during the system operation. An ET approach offers the following advantages

- it lends better flexibility of the communications.

- may result in faster response times for the asynchronous events occurring at unknown times. Depending on the network capacity and the load, at certain points, may show better real-time performance than with time-triggered systems.

- no clock synchronisation or global schedule building is necessary. Thus, it is not necessary to equip network elements (end systems and switches) with specific synchronisation components.

On the downside, event-triggered approaches are subject to

- variable transmission jitter and non-deterministic delays since several messages may arrive at any instant and queue up in switch ports.

An event triggered approach is inherently non-deterministic, however, as we studied, it can achieve a level of determinism using techniques such as shaping or bandwidth reservation. The choice for time-triggered or event-triggered communication greatly depends on the application, as noted in other works [55, 21]. A realistic system will generally contain multiple applications, some of which may communicate using the TT approach while others communicate using an ET approach, for instance, the automotive system. Accordingly, the referred technologies in this chapter inherit the merits as well as demerits of the respective communication paradigm. Table 2.1 provides a quick overview of the different aspects studied above for easy comparison.

2.11 Summary

Over the last years, embedded systems complexity has increased significantly, leading to distribution with high-bandwidth networks, particularly Ethernet and its real-time flavors. In the particular case of the automotive industry, that we used as a motivating example, car technology

has witnessed continuous developments led by emerging requirements such as comfort and safety among others; moving from mechanical components towards electronics being a major shift. This shift has enabled multiple applications in the automotive domain with demanding requirements regarding timeliness and bandwidth. To this end, Ethernet has received a growing interest as a possible network to replace the disparate communication means or converge them in a common backbone.

In this chapter, we have studied AFDX, AVB, TTEthernet and FTT-SE. These technologies support varying QoS. We also compared these technologies along different aspects, and we choose to use FTT-SE in our work. The following points motivate our choice.

As noted earlier, all the mentioned technologies except FTT-SE support limited priority levels, mapping the frame priorities into classes, often employing the FIFO policy between different flows in the switch queues. Thus, some applications may suffer unacceptable delays due to mutual interference between frames of the same class. Moreover, as described in chapter 1, we envision supporting reservations in a multi-level hierarchy, and all the protocols as mentioned earlier do not support such reservations. Some of the studied protocols provide temporal isolation of competing flows using traffic shaping mechanisms, for example, virtual link scheduling within AFDX, scheduling SR class A and B within AVB and rate-constrained traffic scheduling within TTEthernet. These shapers are servers that specify the available bandwidth for associated messages in a given time window. However, these techniques essentially provide a hierarchy that is limited to two levels only; at the top level, system scheduler regulates network access between multiple shapers, and all individual servers are at the level below, controlling transmission of the associated messages. For instance, the sub-VL concept in AFDX offers some similarity to the hierarchical scheduling. However, multiple applications can send messages in the same sub-VL queue which is FIFO, potentially degrading performance for a critical application. Furthermore, messages from different sub-VLs are converged into a VL in a round-robin manner (disregarding priority) and later transmitted in FIFO order respecting BAG interval between successive transmissions. Time-triggered frameworks reserve fixed periodic slots which are in fact polling or periodic servers. These frameworks impose a compromise between response time and bandwidth; more bandwidth must be reserved to obtain shorter response times. However, these frameworks do not allow arbitrary server policies with hierarchical composition. FTT-SE allows changing QoS of the communications dynamically. The work in [102] observes that FTT-SE provides higher flexibility than AVB in this respect. Currently, FTT-SE supports dynamic management of the streams, which is extensible for the servers. Finally, these technologies require specific modifications in the switches, for instance, AVB bridges and TTEthernet switches, making them less general while FTT-SE works on top of COTS Ethernet switches. Nevertheless, it requires all nodes to be FTT-compliant.

Table 2.1: Comparison of different technologies

	Technologies			
	AFDX	AVB/TSN	TTEthernet	FTT-SE*
Priority levels	1 or 2	upto 8	3	arbitrary
Traffic types	ET	TT, ET and BE	TT, ET, BE	TT, ET, NRT
Class queue scheduling policy	FIFO	FIFO	FIFO	EDF, RM, DM, FIFO
Bandwidth reservation	✔	✔	✔	✔
Synchronization	✗	✔	✔	✔
Fault tolerance	✔	✗	✔	✗
Dynamic QoS	✗	✔	✗	✔
Hierarchical scheduling	2 levels	2 levels	2 levels	multiple levels

Chapter 3

Traffic Scheduling Concepts

In Chapter 2, we studied some candidate technologies for efficient communications in CPS, with automotive as an example case. This chapter presents different techniques that we use in system design, for instance, server-based traffic scheduling. Such techniques allow achieving the desired Quality of Service (QoS). QoS refers to the perceived value in the performance of the system and covers corresponding mechanisms that render a differentiated service across different flows, typically realised by managing the interfering traffic in the network. For the real-time systems, QoS is often gauged by the timeliness of their communications. QoS is a well-known concept in the general-purpose networking domain. In this domain, however, timeliness requirements are less stringent, and QoS is often measured by related metrics such as packet loss ratio, or throughput. In this chapter, we also present the related work regarding QoS in networks and real-time Ethernet.

3.1 Server-based Scheduling

In this work, we consider the case of scheduling aperiodic traffic. Traditionally, soft aperiodic messages have been assigned a low priority and were scheduled for the remaining time after periodic traffic had been scheduled. This approach known as *background processing* led to long response times for aperiodic traffic when the system was overloaded. Thus it was not well suited for aperiodic traffic that had stringent timing requirements as the utilisation provided to the aperiodic requests can be very low. Server-based scheduling has been used to solve this problem. Servers have an associated budget per time interval making it possible to enforce a bound on the network bandwidth that aperiodic messages may use. By assigning aperiodic messages to servers, aperiodic traffic can have a desired share of the resource, and its impact on the remaining system can be bounded too. Different server-based scheduling strategies exist that vary regarding average response time for associated applications. We describe some typical server policies here. Note that, since we will resort to common scheduling techniques applied on the processor resource, we will mostly refer to *tasks*. However, these scheduling techniques are equally applicable to recurrent messages transmitted over the network resource.

3.1.1 Polling server

Polling Server (PS) is a periodic task created to serve the aperiodic requests. A polling server can be represented by the model (C_s, T_s) where C_s is the budget and T_s is the period of the server task. At regular intervals equal to its period T_s, the server becomes ready and serves pending aperiodic requests within the limit of its capacity C_s. If no requests are pending when the server becomes ready, then it discards its budget immediately and suspends. Also, the server does not retain any budget when the queue of aperiodic requests becomes empty. The server is replenished periodically with full capacity.

3.1.1.1 Example

We consider an example task system given in Table 3.1 and observe the response time of aperiodic requests when served in the background (Figure 3.1) or with a polling server (Figure 3.2). We assign priorities to the task set using RM scheduling policy [103]. The polling server task τ_s is given by the model $(1,3)$ and runs with the highest priority under RM scheduling policy.[1] Figures show the task execution from time $t = 0$ until $t = 30$. Aperiodic requests arrive at $t = 5$, $t = 12$ and $t = 22$ with demands of 1, 2, and 1 units of time respectively. Figure 3.1 shows that response time of aperiodic requests with background service is poor. The service cannot begin until the periodic tasks are completed. The response time of the three requests is 3, 7 and 6 respectively while they need a maximum of 2 units to complete. Figure 3.2 illustrates the same example of aperiodic service obtained through polling server. At time $t = 0$, all tasks are ready, τ_s takes the precedence, but there are no pending aperiodic requests. Thus it discards its capacity. At $t = 5$, a request for 1 units arrives. τ_s schedules the request when it becomes ready at $t = 6$. The request arriving at $t = 12$ is partially served at $t = 12$ and then it completes at $t = 15$. The response time of aperiodic requests with the polling server is better, i.e., 2, 5 and 3 respectively. Thus we see that a polling server shows improvement over background service if the server has a priority that is higher than the lowest in the system. However, it is not always able to serve the requests immediately.

Table 3.1: A task system with two periodic tasks and a polling server

	Execution Time	Period	Priority
τ_a	2	4	Intermediate
τ_b	3	10	Low
τ_s	1	3	High

3.1.2 Deferrable server

Similarly to the polling server, a Deferrable Server (DS) is used to serve aperiodic requests and we represent it with the model (C_s, T_s) where C_s is the budget and T_s is the period of the server. When

[1] Total utilization of the task system in this example is > 1, i.e., $\frac{2}{4} + \frac{3}{10} + \frac{1}{3} = \frac{68}{60} > 1$. Thus, the system is not schedulable. However, this example serves to explain the operation of the polling server and its comparsion to the background service.

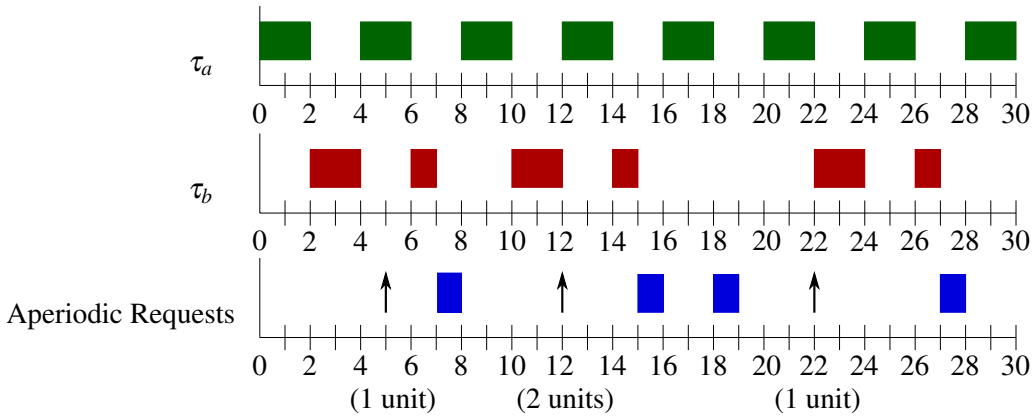


Figure 3.1: Background service

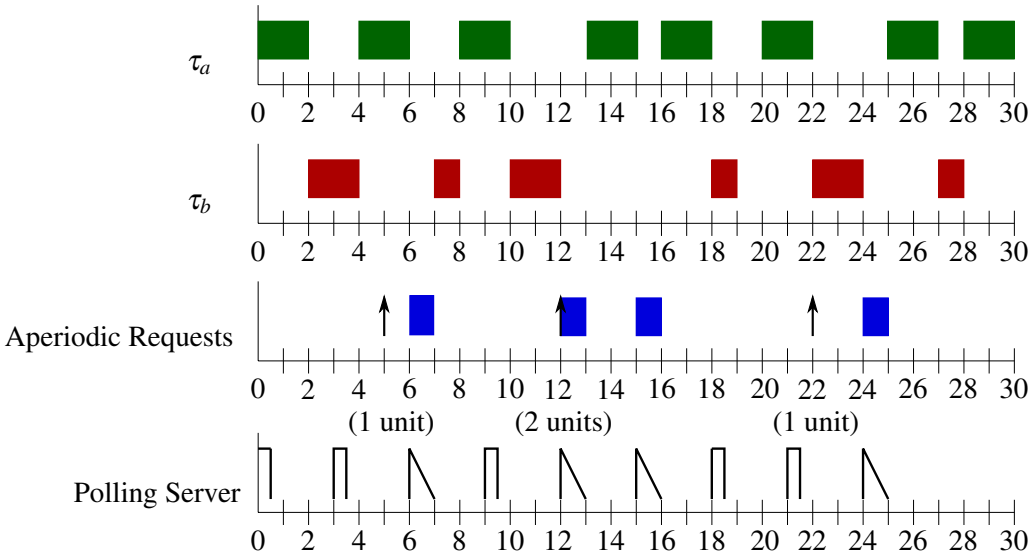


Figure 3.2: Polling service

the server executes (serving aperiodic jobs), the budget is consumed. Unused budget is retained throughout the server period. Thus, even when a job misses a polling instant (upon the invocation of the server), it can be served later in the period. In this way, deferrable server improves the responsiveness of the aperiodic requests. The budget is replenished to C_s at multiples of T_s, but it is not carried over from one period to the next.

3.1.2.1 Example

We illustrate the operation of DS through an example in Figure 3.3, which we borrowed from [104, p. 131]. The task and server parameters are given in Table 3.2. The server runs at the highest priority. Note that server replenishes its budget periodically. Also, since the server preserves its remaining budget during the period, the request arriving at time $t = 9$ can be served immediately. This request preempts task τ_1 since $T_s < T_1$. When the third request arrives at time $t = 11$, the server capacity is zero; hence, its service is delayed until the next period. The fourth request arrives at $t = 16$, but it has to wait until $t = 18$ to get service.

Table 3.2: A task system with two periodic tasks and a deferrable server

	Execution Time	Period	Priority
τ_1	2	8	Intermediate
τ_2	3	10	Low
τ_s	2	6	High

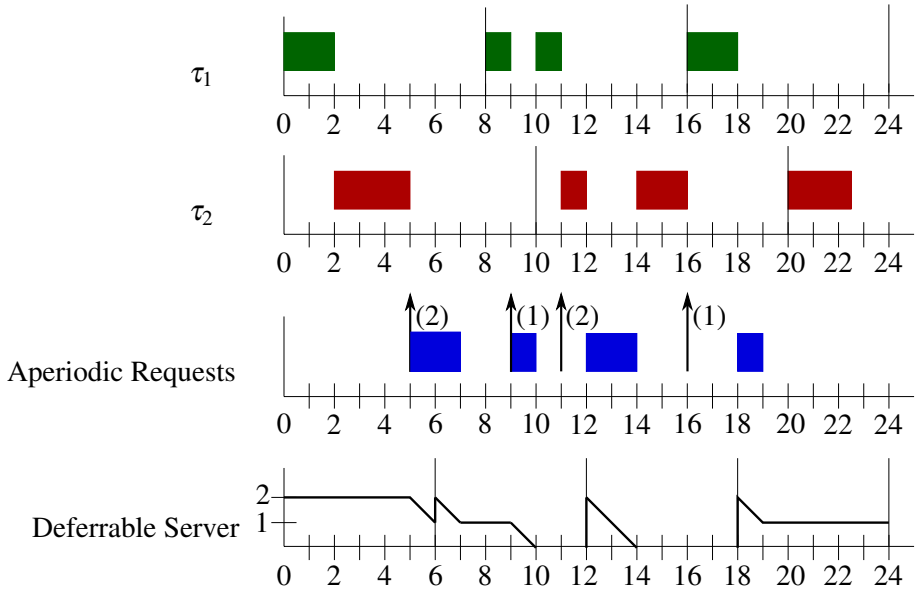


Figure 3.3: An example of a high priority Deferrable Server

3.1.3 Sporadic server

Another kind of server is the Sporadic Server (SS) that we represent with the model (C_s, T_s) where C_s is the budget and T_s is the replenishment time of the server. Consider a request for task τ_1 that arrives at time $t_{arrival-\tau_1}$. The server sets a replenishment instant for itself that is T_s after the task arrival time i.e., the next server replenishment shall take place at $t_{arrival-\tau_1} + T_s$ if the server still has some capacity left, else, the replenishment is delayed to T_s after the previously

scheduled replenishment (i.e., as soon as the capacity becomes greater than zero). Let us say that task requests k time units from the sporadic server. The server budget is decremented to $C_s - k$ units. At time $t_{arrival-\tau_1} + T_s$, k time units are returned to the current server capacity. If a request for task τ_2 arrives soon after, at time $t_{arrival-\tau_2}$ needing j units from the sporadic server, its capacity will be decremented to $(C_s - k) - j$ units, and the next server replenishment will take place at $t_{arrival-\tau_2} + T_s$ for the consumed j units (Figure 3.4). In Figure 3.4, notice the aperiodic request that occurs at time $t = 7$ when there is no available server capacity. Therefore, the next replenishment instant is set T_s time units after the instant that capacity becomes greater than zero. In our example, this is the replenishment instant occurring at $t = 8$, and thus, replenishment for the request arriving at $t = 7$ is due at $t = 15$. Thus, a sporadic server enforces a bandwidth equal to C_s/T_s and its worst-case impact on the rest of the system equals that of a periodic task with similar parameters [104, p. 147]. Note that a sporadic server may serve a request immediately depending on its relative priority and on whether it has enough capacity. When the capacity is exhausted, the service is suspended until the next replenishment.

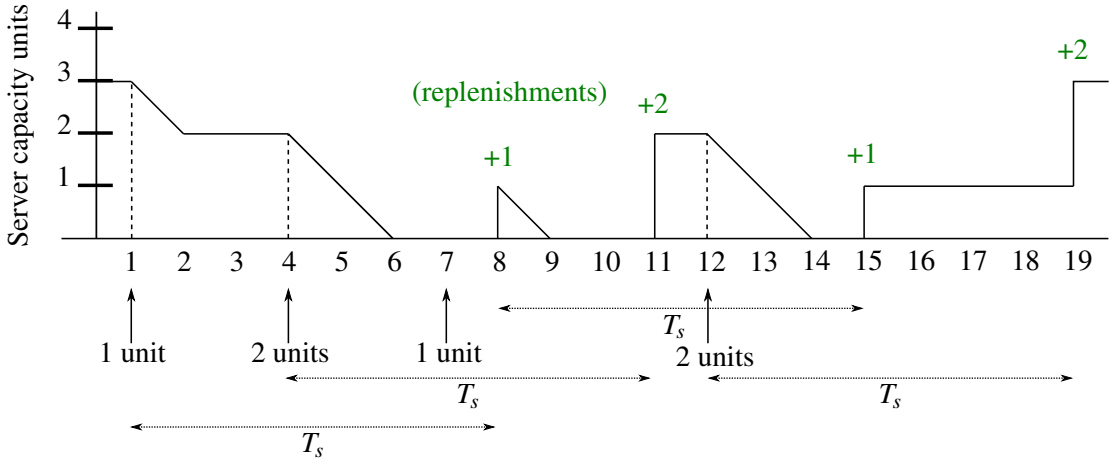


Figure 3.4: The sporadic server with $C_s = 3$ time units and $T_s = 7$ time units.

However, a sporadic server has more complex rules regarding its scheduling and replenishment scenarios. We cover these aspects for the sake of completeness through an example which we reproduce from [105]. This example highlights a case when a low priority task in a schedulable system misses its deadline due to an effect known as *premature replenishment*.

3.1.3.1 Example

Consider a task system with two tasks and sporadic server that shall handle apeirodic requests (Table 3.3). The priority is defined using deadline monotonic policy [106].

First, we check if this task set with sporadic server is schedulable. To determine schedulability, we check whether the server and tasks meet their deadlines upon critical instant (released simultanesouly at time $t = 0$.) We find that worst case response time for τ_1 is $10 < D_1$, for τ_2 is $99 < D_2$ and for τ_s is $30 < D_s$, and therefore tasks and the server are schedulable (Figure 3.5).

Table 3.3: A task system with two tasks and a sporadic server

	Execution Time	Period	Deadline	Priority
τ_1	10	200	20	High
τ_2	49	200	100	Low
τ_s	20	50	50	Intermediate

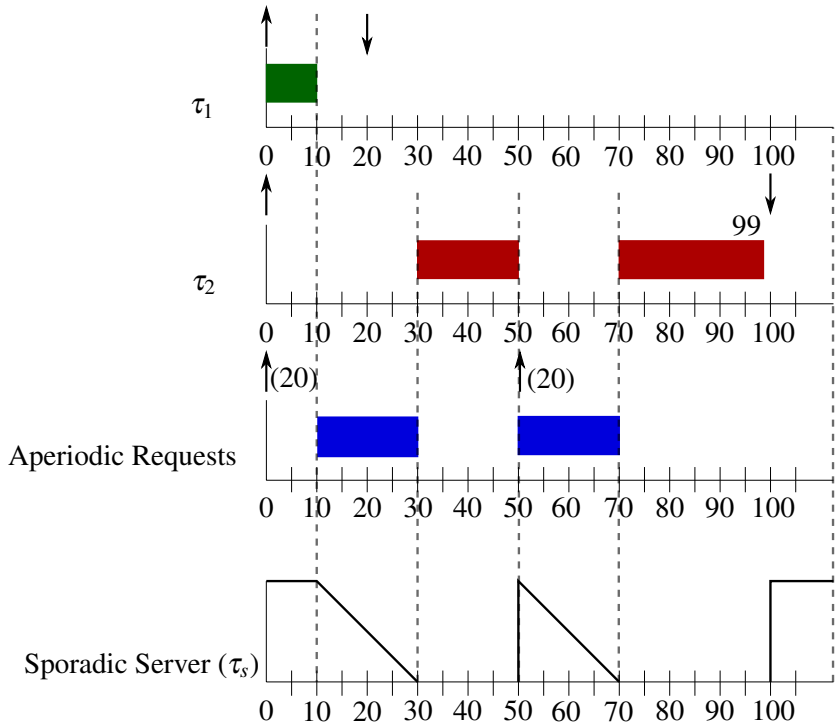


Figure 3.5: Critical instant of the system with sporadic server

Next, we consider another scenario where task τ_1 arrives at time $t = 41$, and sporadic server τ_s and task τ_2 are simultaneously released at time $t = 0$. Moreover, three aperiodic jobs arrive at time $t = 0$ requesting 18 units, at time $t = 40$ requesting 20 units and at time $t = 90$ requesting 20 units. Figures 3.6 and 3.7 depict this situation.

In this example sporadic server is executed at medium priority, depicting also the case where a task misses its deadline. This happens due to the fact that an amount greater than server initial budget may be accumulated through replenishments within a server period. In this particular example, it allows three aperiodic requests to be issued which result in task τ_2 missing its deadline with response time of 117. Therefore, a sporadic server deserves careful consideration (refer [105]).

3.2 Hierarchical Scheduling

Hierarchical scheduling has received growing attention in the design of complex distributed embedded systems with real-time requirements. The early work in [107] presents a two-level hierar-

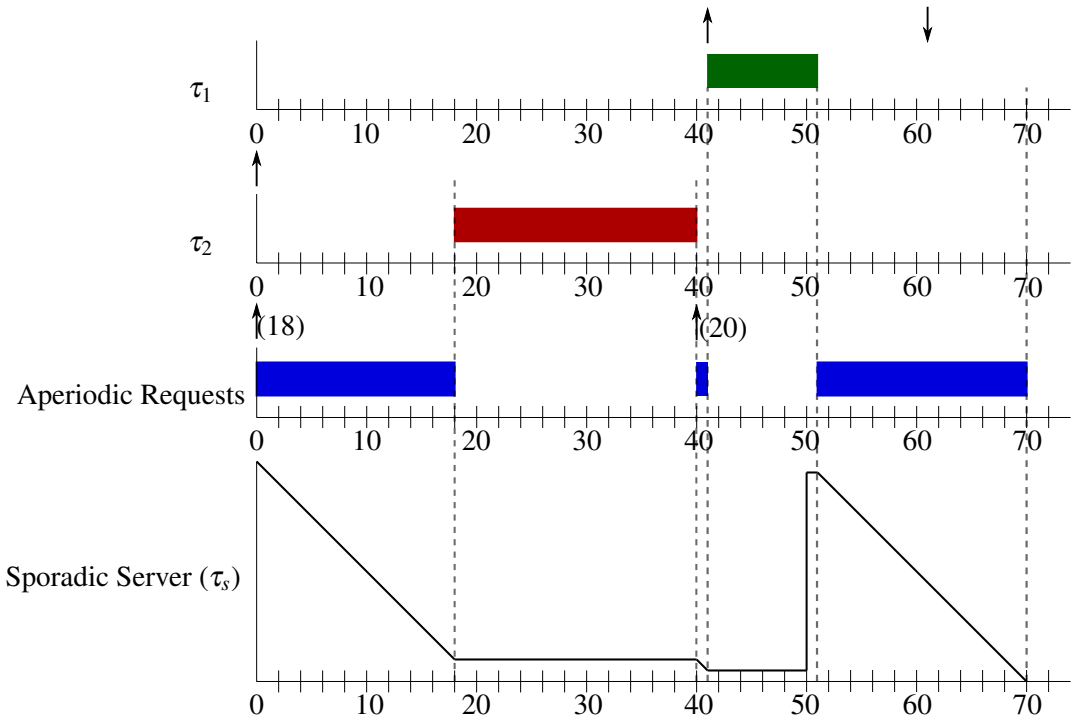


Figure 3.6: A complex scenario of message scheduling with sporadic server (continued in Figure 3.7)

chical scheduler to schedule a system consisting of real-time and non-real-time applications. The idea was to devise a method that could analyse the tasks in the real-time applications in isolation and later could integrate the applications while ascertaining system schedulability. Their approach creates a server with constant utilisation for each real-time application, and a single server manages all non real-time applications. At the top level, the operating system schedules the servers according to Earliest Deadline First (EDF) scheduling strategy. On the second level, the selected server can apply any application-specific scheduling algorithm such as RM, EDF, Time Sharing schedulers, to schedule a set of application tasks.

The concept of *open systems* was introduced in their work [108] to develop schedulability analysis for the cases when an independently developed application needs integration into a system which has been previously analysed without this application. Most analyses in real-time systems perform global schedulability analysis for *closed systems* in which all applications are known a priori and are analysed together. For an open system, the decision to accept a new application is based on whether the new application and the currently executing applications remain schedulable after the new application is included in the system. This method can, however, be inapplicable if the characteristics of the applications are unknown. The work in [108] builds on the two-level scheduling hierarchy presented in [107] and extends this work to overcome certain limitations,

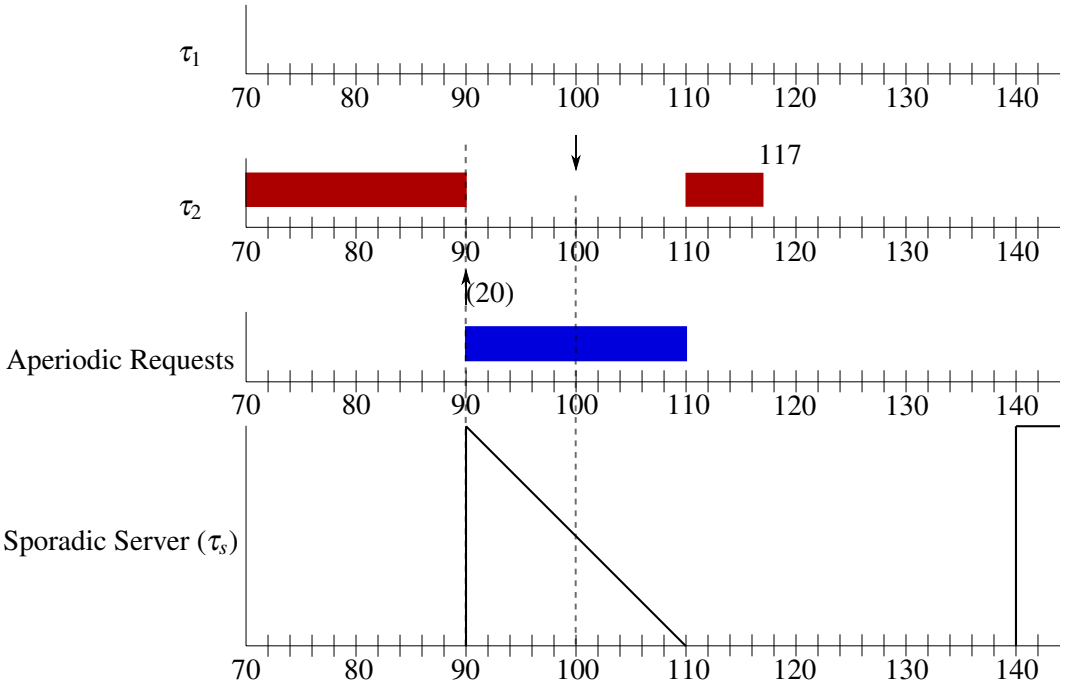


Figure 3.7: A complex scenario of message scheduling with sporadic server. τ_2 misses its deadline (continued from Figure 3.6).

such as non-preemptive execution of applications and the impossibility to share resources across different applications.

Limited forms of hierarchical scheduling, typically with two levels, only, have been used for many years in the networking domain to partition the network in virtual channels and to bound the burstiness of certain types of traffic. For example, traffic shapers are one form of servers that enforce some level of hierarchical scheduling, limiting the amount of traffic a node can submit to the network in a given time window. The shaper has one scheduling discipline associated with its arrival queue, typically FCFS, and another scheduling discipline manages its output at a global level. The leaky bucket is a common traffic server frequently found in networks that belong to the category of shapers [109]. In general, network servers use techniques similar to those of CPU servers, based on capacity (or budget) that is eventually replenished. Many different replenishment policies are also possible, being the periodic replenishment as with the Polling Server (PS) or the Deferrable Server (DS), the most common ones. However, it is hard to categorise these network servers similarly to the CPU servers because networks seldom use clear fixed or dynamic priority traffic management schemes. In fact, there is a large variability of Medium Access Control (MAC) protocols, some of them mixing different schemes such as round-robin scheduling, first-come first-served, few priority queues, etc.

Many works in the literature address the issue of hierarchical scheduling and component-based design [110, 111, 112, 113, 114]. The work in [110] addresses both the analysis and the design problem, i.e., given a set of tasks and a fully preemptive scheduler, it tests the schedulability

of a task set with given server parameters and later designs the server to utilise the least system resources while making the workload schedulable. The underlying server model is periodic, and their notion of server availability function is similar to that of resource supply in [112]. The design approaches in [110], [113] and [111] consider two levels of hierarchy only. The server interface composition towards multiple levels is thus not covered. We argue that a multilevel hierarchy supports a more complex application composition and bandwidth distribution. The work in [113] reports an increase in the remaining utilisation when server periods are exact divisors of the task periods and task arrival times coincide with replenishment of the server capacity; such tasks are called *bound* tasks in their work. The work in [115] presents an end-to-end resource reservation mechanism taking into account both the computing and the communication resources. This work also includes an accompanying analysis and a design method for the reservations.
In our work, we will use the knowledge of application parameters to design server interfaces; however, application tasks can be released independently of the servers. The work in [116] develops an incremental schedulability analysis that considers a periodic resource supply model. In their approach, a component must export its worst-case resource demand depending on the task model and scheduler. None of these works considers the specific case of polling server [26] within an HSF. In general, all the previous approaches consider a general fully preemptive system/application level task scheduler. Therefore, we considered two important aspects while designing our approach; one is the non-preemptive nature of packet scheduling, and the other is the use of polling server policy.

3.3 Guaranteeing Quality of Service

Over the years, a number of distributed resource management strategies were developed that handle multiple heterogeneous resources, particularly CPU and network, occasionally integrating disk, memory and energy, creating partitions and providing consistent reservations. There are systems that must support applications having varying QoS requirements in terms of timeliness, reliability or security. The work in [117] builds a model to manage QoS along such different dimensions and is known as QoS based Resource Allocation Model (Q-RAM). Applications may specify their resource demand within a range (i.e., minimal and maximal resource requirements). The model assumes that the resources are sufficient to meet the lower bound of resource demand from each application even when the system cannot satisfy the maximal demand requested by applications simultaneously. They use the concept of utility function that computes the application's total utilization of all the resources. The application is feasible if it can be allocated the minimum set of resources along each QoS dimension. The objective is to maximize the system utility while guaranteeing that each application is feasible along each of its QoS dimensions. Relative importance of an application is taken into account while computing total system utility.
The work in [118] reports an implementation of the Q-RAM model to a radar system. A radar's

function is to track targets. This function is achieved by transmitting a beam and receiving the return echoes, both in a non-preemptive fashion. Such systems change rapidly since the targets move continually. Therefore, resource allocation and scheduling needs to be done frequently and in real-time. The resources considered are time and energy. The problem becomes that of multi-resource allocation and real-time scheduling. Using the Q-RAM model, the work aims to maximize the total utility derived by tasks in the system while meeting the resource constraints.
The work in [119] considers network based reservations of heterogeneous resources and strives to provide end-to-end QoS. For such reservations, communication may span geographical networks that reside in different administrative domains and thus have different control policies. To achieve QoS, required resources need to be configured, reserved and allocated. The resources they consider may include computers, networks, disk and memory. They present Globus Architecture for Reservation and Allocation (GARA) that is based on the Globus tool-kit. This architecture allows management of computational elements or resources and their reservations through co-reservation and co-allocation agents that can have centralized or distributed implementations. The Globus architecture has three main components, an information service, local resource managers and co-allocation agents. Information service is based on Lightweight Directory Access Protocol (LDAP) and structures the resources hierarchically. The entries within the information service include information on resource attributes such as type, architecture, current state etc. To allocate a resource, the application passes its requirements to the co-allocation agent. This agent, on behalf of the application, uses a combination of queries to the information service, heuristics, and application-specific knowledge to map application QoS requirements onto resource requirements, and then discovers resources with those requirements, and allocates the resources. Within GARA resource objects can encompass resources of different types such as network flows, memory, disk etc as well as it is possible to make reservations in advance. The Globus system uses an exhaustive search to choose the desired resources. Through a set of instructions, they create reservations and allocate resources and carry out the management of these resources. However, the work does not detail how the application semantic values are transposed to resource requirements. They consider a large geographical area, and latency incurred in resource allocation may not be suitable for applications with hard real-time requirements. Moreover, there is a possibility of application being blocked waiting until all the required resources are available i.e., in situations where a different co-allocation agent might be holding a needed resource.

3.3.1 QoS at network layer

In the case of networks with multiple networking components, such as switches in a LAN, it is common to find FIFO queues that implement a best effort service model. With this model, all packets receive the same quality of service. Under light load conditions, such quality can be sufficient; however, applications may receive poor service when the network is heavily loaded. When there exist multiple applications with differing requirements, e.g., email with a low-bandwidth requirement, HDTV with high bandwidth requirement, and time-sensitive audio conferencing, best-effort

Internet service model may be unsuitable. One solution is that applications specify their service needs and the network reserves resources to meet these needs. To meet such aims, the early works in [10, 11] advocate more efficient than FIFO packet scheduling algorithms and the idea of allocating resources selectively. The underlying motivation is that packet scheduling greatly impacts the resource allocation. This work is the basis for the Internet *integrated services model*. An admission control mechanism decides which flows can be accepted to support quality of service. With such admission control in place, a scheduling algorithm is effective since it can keep the aggregate traffic load to the level where service commitments are enforced. In particular two types of service commitments have been considered in this work; QoS commitment to individual flows, and resource sharing commitments to collective entities. For the first case, quantitative service commitments can be provided ensuring that the network will meet or exceed the contracted QoS. QoS, in turn, is defined by a metric. An example of quantitative service commitment is to meet the bound on maximum packet delay. An application's usual behaviour can determine the value of such a metric. The value is a threshold beyond which the application performance will degrade significantly. Relative service commitment assures how packets of one flow shall be treated relative to the other flows. Priority ordering is an example of this kind of service commitment. This work contends that per-packet delay is a fundamental measure defining QoS that an application receives, and hence bounds on the minimum and maximum packet delay are of particular interest. To discuss the case of real-time applications, audio-video live streaming applications are frequently taken as examples. For such applications, network-induced jitter distorts the received audio-video. To be able to reproduce the source signal as faithfully as possible, the receiver may buffer the incoming packets after they arrive at the destination for a duration such that the playback point of the packet is at some maximum offset delay from its original departure time. The data that arrives after the playback point can be discarded. Two important dimensions in this respect are the latency and fidelity. Latency is the same as the maximum offset delay that is chosen to playback the signals. Fidelity is how closely the received signals match the original. Applications that need a very high degree of fidelity known as intolerant choose the fixed offset value greater than the absolute maximum delay of packets to avoid the possibility of late packets and hence the resulting signal distortion. Applications receive a *guaranteed service* if a reliable upper bound to the delay is given. Tolerant applications, on the other hand, do not need to specify a delay value that is greater than the absolute maximum as they can tolerate some late packets. Frequently, these applications can adjust the delay value they tolerate based on the experience of recent packets received. In this way, it is possible to reduce the impact of the network latency on the application performance.

In the Integrated Services Framework, the two classes of tolerant and intolerant applications can be loosely mapped to soft and hard real-time applications as used in common literature for real-time systems. In this framework, intolerant applications can be handled with a *guaranteed service model* whereas the tolerant applications can be managed with a *controlled service model*. Hard guarantees are provided for the intolerant applications only.

Whenever a link is to be shared among a collection of applications, in which QoS to individual

flows must still be guaranteed with adequate delay bounds, it is important to share the link in a controlled manner to protect against overload situations. For this purpose, the Integrated Services Framework proposes using reservations that can be enforced over the Internet Protocol with the Resource Reservation Protocol (RSVP) [120]. This protocol establishes reservations along a communication route for a TCP/UDP communication stream. The creation of reservations includes a negotiation phase for reserving resources in every router along the respective route according to the so-called FLOWSPEC of the respective stream. Nevertheless, more efficient network reservations, mainly under tight time constraints, require reservations that are supported on the data link level.

3.3.2 Scheduling and QoS in Ethernet

Guaranteeing QoS on Ethernet is typically achieved with the so-called Real-Time Ethernet (RTE) protocols, which use some limited forms of server-based traffic scheduling. Some protocols enforce periodic communication cycles with reserved windows for different traffic classes (e.g., AVB [22], TTEthernet [25, 121, 122] and AFDX [62]). This is a trivial composition of several PS that hardly support efficient use of the network bandwidth. In the context of switched Ethernet, some works consider traffic shaping at the end nodes such that submitted traffic conforms to some average rate or maximum burstiness that switches can handle [109]. In this respect, the same technologies (e.g., TTEthernet [25, 121, 122] and AFDX [62]) offer some traffic shaping mechanisms in the end nodes, as we studied in Chapter 2.

However, due to infrastructural limitations, none of these protocols supports arbitrary server policies nor their hierarchical composition and dynamic adaptation or creation/removal. A step further is given by Linux-TC (Traffic Control) [123], which provides a reconfigurable hierarchy of a wide choice of server policies and traffic filtering at each node. However, it lacks a global coordination scheme to enforce consistent management of the server hierarchies across the distributed system.

Focusing on Ethernet technology, several works attempted to provide resource partitioning and reservations dynamically. EtheReal [124] was probably the first such protocol that used a tailored switch that provided guaranteed-bandwidth network services based on explicit resource reservations. A connection set-up triggered a message that was sent back and forth along the desired path, reserving the needed resources in all hops. Other examples include the special switch of Hoang et al. [125] that forwarded a mix of real-time and standard IP traffic using Earliest Deadline First (EDF) policy, with timeliness guarantees provided by adequate on-line admission control in the switch and in the end nodes, or, more recently, the hard real-time communication over multi-switch networks using ordinary Ethernet hardware proposed by Zhang et al. [126], just making use of a dual-level traffic smoothing mechanism analysable by Network Calculus.

In general, solutions that provide real-time over Ethernet require specific modifications or support in the hardware which incurs high deployment cost and limits their experimental use for possible extension or further research. The work in [127] presents *Atacama*, a hardware-based, open framework that provides real-time communications in multi-hop switched Ethernet networks. End

stations with real-time data use Application Specific Instruction-set Processor (ASIP) which enables defining and executing time-triggered schedules. Ethernet interface includes an arbiter to differentiate the time-sensitive traffic from the best effort traffic. Real-time traffic is always prioritised, reducing the bandwidth available to the best-effort traffic. Moreover, within the switch, real-time frames are forwarded in a cut-through fashion thus reducing latency and jitter. Finally, the recent AVB standard [22] uses SRP (IEEE 802.1Qat) [65], which is a two-stage (registration and reservation) method for dynamic bandwidth reservation. Moreover, reserved channels are shaped at the output link using FQTSS (IEEE 802.1Qav) [66]. The source nodes are expected to be shaped, too, e.g., using a leaky bucket.
Looking back at all those protocols, they either provide rather limited priority levels (AVB) and/or miss the semantic connection between streams of the same application (all), and/or make fixed reservations along the streams paths without considering the load of each switch (all but Ethereal). We attempt to resolve these limitations providing one single network reservations protocol that caters to the needs of efficient CPS.

3.3.3 Scheduling in FTT-SE

Concerning FTT-SE, the work in [128] presented a proof-of-concept implementation of servers within FTT-SE that was called Server-SE. However, this preliminary work missed a full perspective regarding hierarchical scheduling deployment and system analysis. The work in [99] presents a similar server deployment approach but inside an Ethernet switch, while its analysis appears in [129].
The work presented in this thesis refers to the following recent projects:

- the HaRTES project [98, 99], within which an FTT-enabled Ethernet switch was designed and built that supports dynamic and adaptive isochronous and asynchronous channels with temporal guarantees and mutual isolation;
- the Serv-CPS project [130] which is a continuation of the HaRTES project and which provides dynamic hierarchical reservations within the FTT-enabled switch.

An early work exploring the scaling of FTT-SE [31] network is reported in [131] while a load-aware resource reservation in FTT networks for isochronous channels was proposed in [132]. An analysis of worst-case response time was developed either for single resources (each switch) [43] as well as for multiple switches [44].
Mohammad et. el explored the delay analysis for multi-switch FTT-SE networks and multi-HaRTES networks. The work in [133] presents a delay analysis based on network-calculus formalism for one particular scalable architecture based on FTT-SE. This multi-switch architecture comprises multiple masters where a distinct master controls part of the network referred to as the *sub-network*. This architecture entails design adaptations in the scheduling model such as additional partitions of the EC to schedule local and global traffic as well as multiple trigger messages

for synchronisation among different masters. On an example network, an evaluation of this analysis reports delay bounds that are up to 50% larger than with a corresponding simulation of the protocol scheduler.
The work in [134] presents a multi-hop HaRTES architecture with an accompanying traffic forwarding method called Distributed Global Scheduling (DGS). With this approach, the messages that traverse several switches are buffered in each switch except the last switch en route, where they are forwarded to the destination without being buffered. Hence, for a synchronous message, a phase is defined in each switch on its route, which is the delay (in number of ECs), that the message suffers from its activation in the source node to its scheduled time in that switch. A delay analysis based on request bound and supply bound functions is presented for a single HaRTES network and extended for multi-switches. The introduced phase in the referred traffic forwarding method is not accounted for in the analysis adding to its pessimism.
The work in [135] proposes an improvement for traffic forwarding in a multi-hop HaRTES network. Depending on the characteristics of the traffic, transmission window sizes can be different in different links to improve the utilisation of available bandwidth. This method called Reduced Buffering Scheme (RBS) modifies switch output queues to prioritise the traffic and avoids buffering incoming messages as long as they can fit within the transmission window on an outgoing link. An iterative response time analysis is developed that accounts for the idle times in transmission windows, considering the worst scenario when windows sizes are different between two links. The two methods (RBS and DGS) are compared by evaluating their performance against two example networks. For high priority messages RBS always performs better, for medium or lower priority traffic, it performs better or equally in most cases.
The proposed work in this thesis is a continuation of these recent efforts towards supporting efficient CPS using FTT-SE as real-time Ethernet technology. In particular, our work focuses on leveraging multi-level hierarchical scheduling to multiplex single-switch networks.

3.4 Summary

This chapter outlined server-based scheduling as a technique to achieve QoS in real-time networks and presented some related work regarding hierarchical scheduling and QoS. QoS within the Internet, in particular, at layer 3 and above, has been explored in the Integrated Services Framework, where, the feasibility for enabling real-time applications was correlated with the performance of the packet scheduling mechanisms inside switch queues.
In full-duplex Ethernet, QoS is achieved by managing the interfering traffic, inside the nodes and switches. This, in turn, may be realised by different mechanisms, such as priority assignments to the applications, traffic shaping policies, or by constructing time-triggered schedules where fixed slots are reserved for different classes of traffic. In this chapter, we saw how the server-based scheduling techniques could be used to deploy traffic shaping policies or even bandwidth reservations and thus achieve adequate QoS, particularly concerning timeliness. Moreover, we also saw that real-time Ethernet protocols lack the support for multi-level hierarchies. Finally, this chapter

presented some lines of work carried out within FTT-based architectures including scalability of FTT-SE, server-based scheduling within HaRTES, and multi-HaRTES architecture. The work presented in this thesis will provide multi-level hierarchical scheduling within single-switch Ethernet networks to support the design of efficient CPS.

Chapter 4

Analyzing the Efficiency of Sporadic Reservations on Ethernet with FTT-SE

Resource reservations, e.g., processor or network, can be enforced with server-based scheduling, possibly hierarchical. Reservations are an effective way of isolating different applications or their parts and thus, support composability and design of complex systems. However, when reservations are not fully used, they become a potential source of resource inefficiency. This is particularly relevant when reservations are designed according to worst-case requirements. In this chapter, we analyze the efficiency of a specific worst-case network delay analysis for reservations over Ethernet, taking the pessimism of the analytic worst-case delay as an indirect metric of potential bandwidth efficiency. We focus on Ethernet due to its growing relevance in the distributed embedded systems realm and we use the Flexible Time-Triggered Switched Ethernet protocol (FTT-SE) [31] that allows any traffic scheduling policy, including hierarchical reservations. In this chapter, we consider reservations associated with individual asynchronous messages that enforce a minimum inter-transmission time. These reservations are simplified sporadic servers that deplete their capacity upon each invocation and which we designate as *flat servers*.

The work presented in this chapter is built around [37] that provides an analytic model for the worst-case response time of asynchronous messages within FTT-SE and [38] that gives a preliminary efficiency assessment with random message sets. We further assess the efficiency of this analysis using extensive simulation runs of the FTT-SE master scheduler and comparing the analytic delay upper bounds with the observed maximum message response times under different system configurations. Namely, we vary properties of message set, network configuration parameters and protocol configuration parameters. Considering schedulable sets, our target is to understand how the tightness of the analysis behaves. Tightness refers to how close is the analytical estimate of message response time to the worst-case response time observations obtained from the protocol scheduler within a given simulation trace. Our empirical findings can tell us which configurations favor the analysis more. This knowledge can help system designers better tuning their designs to improve network efficiency under strict timing guarantees. We took inspiration from a similar effort developed in the past for processor scheduling, namely the Hartstone benchmark [136].

Despite the existence of worst-case delay analysis for most of the protocols referred in the previous chapters, to the best of the authors' knowledge there is no wide study on their efficiency. Even if some works discriminate analysis that are more efficient than other, they do not compare with actual system runs. Such study is the aim of this work for the specific case of the asynchronous traffic in FTT-SE. The analysis model present in this work derives from the analysis of synchronous streams on a multi-switch FTT-SE network [44] being adapted to the case of a single switch network and asynchronous streams protected with flat servers. In the remainder of the chapter, Section 4.1 presents the operation of flat servers within FTT-SE. Section 4.2 describes the system model and the response time analysis. In Section 4.3, we present the experimental evaluation comparing analytic model and the FTT-SE scheduler execution. In Section 4.4, we provide some guidelines for system design based on the lessons learned through this study. Finally, a summary of the chapter is given in Section 4.5.

4.1 Flat Servers within FTT-SE

FTT-SE schedules traffic in elementary cycles (EC) ensuring that traffic scheduled for one EC can be transmitted in that EC thus avoiding the backlog in switch queue from EC to EC. This guarantee is given by the schedule building which takes into account that the latest finishing instant for each message in each link does not extend past the configured transmission window. Thus there will be no messages left in the switch queues at the end of the EC [137]. The online scheduling of each traffic class, namely synchronous and asynchronous, takes into account the size of the respective reservations and is given in [31]. Within the reservation for asynchronous message streams, each message is further associated to a simplified sporadic server that depletes its capacity in each invocation (Figure 4.1). Such servers enforce a minimum inter-transmission time in the respective streams and thus protect the rest of the system. We designate them as *flat servers* because they are not meant to support further hierarchical decomposition.

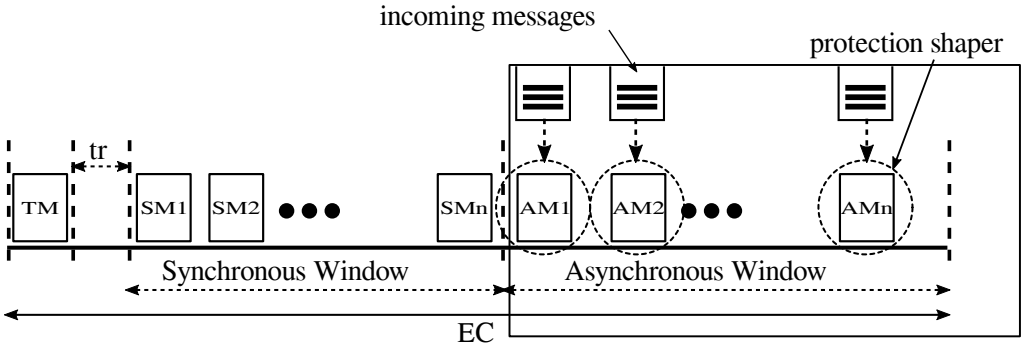


Figure 4.1: Reservations for individual message streams within the asynchronous window

4.1.1 Operation of flat servers

Consider the three main system components in FTT-SE, i.e., the system database, ready queue, and the scheduler. The record for each message is permanently available in the database during the system execution. The global queue of the ready messages contains pointers to the message items. In each cycle, the scheduler consults ready queue for selecting messages for transmission. A message reference becomes available in the ready queue upon its activation and it is erased once that message is scheduled for transmission (Figure 4.2).

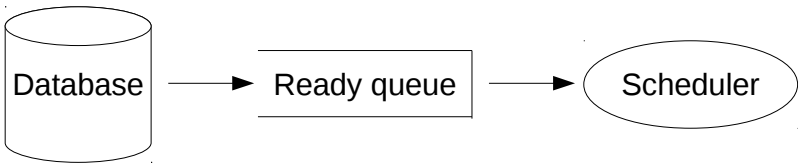


Figure 4.2: Main system components

Associated with a message item there are two variables, namely *activation counter* and *replenishment counter* denoted henceforth by act_ctr and rep_ctr, respectively, that are part of the item's dynamic data and are used to manage the protection mechanism of flat servers, i.e., the minimum inter-transmission time.

Within the scheduling loop, an item selected from the queue is eligible for scheduling (insertion in the EC) if its act_ctr is 0. Otherwise, the scheduler skips the respective item and inspects the following one. At the system start, act_ctr and rep_ctr values are 0 for all items. Hence, in the beginning, each message is eligible to be included in the EC.

During the system execution, a message can be in either active or inactive state. Active messages are present in the ready queue. Inactive messages are present in the database since they are not ready for transmission, yet. In a given EC, active messages can be classified as fully scheduled or partially scheduled, only (note that FTT-SE considers a multi-packet message model). There are two cases concerning the update of the dynamic data of each message:

4.1.1.1 Case A

When an active message is fully scheduled in a given EC (i.e., finishes transmission in that EC), its record is erased from the queue (i.e., becomes inactive) and $\mathsf{act\_ctr} = T + \mathsf{rep\_ctr}$ and $\mathsf{rep\_ctr} = 0$, with T being the message minimum inter-transmission time or period for simplicity. Then, at the end of the scheduling loop, all inactive messages do a count down to their next earliest activation ECs: $\mathsf{act\_ctr} = \max(0, \mathsf{act\_ctr} - 1)$ and $\mathsf{rep\_ctr} = 0$.

4.1.1.2 Case B

When an active message is partially scheduled (i.e., will not finish transmission in that EC either because it is too long or due to interference of higher priority messages), just rep_ctr is decremented at the end of the scheduling loop (i.e., $\mathsf{rep\_ctr} = \mathsf{rep\_ctr} - 1$), keeping track of the cycles

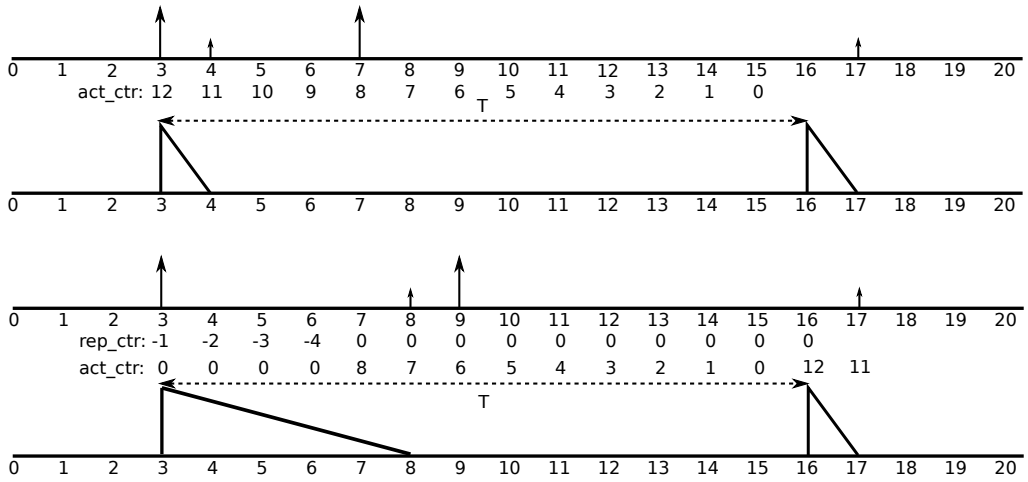


Figure 4.3: Operation of the shapers

for which a message remained in the queue waiting to be fully scheduled. Note that act_ctr=0 during the time in which a message is in Case B. Thus, when a message enters in Case A, the minimum time until that message can be activated again is reduced by rep_ctr (note it is a negative value).

Therefore, when a message request arrives, the respective message can only become active if both act_ctr=0 and rep_ctr=0. Else, it remains inactive until that condition is fulfilled. This is explained in the example of Figure 4.3 referring to a message with the model $(C,T) = (4packets, 13ECs)$. The tall arrow represents a new message arrival whereas a short arrow represents the EC when a message finishes transmission. The figure depicts two situations in which the referred message takes different times to be transmitted. In the upper situation the transmission is completed in 1 EC (Case A). When a second message instance arrives at $t = 7$ it cannot be immediately activated since act_ctr is not 0, but only at $t = 16$ thereby separating consecutive activations by at least T. In the lower situation, the message transmission takes 5 ECs during which rep_ctr keeps decrementing (Case B from $t = 3$ to $t = 6$). At $t = 7$ it enters Case A and starts the countdown with act_ctr. Again, the instance arriving at $t = 9$ will be kept inactive until $t = 16$.

4.2 System Model

In this section, we describe the network and traffic model, followed by the response time analysis of the traffic in the FTT-SE flat servers.

4.2.1 Network model

The switches are considered to be full-duplex where the input and output of a switch port are isolated. Thus, the reception of a message does not influence on the transmission of a message. In this work, we consider the store-and-forward type of switches in which a message is fully

received before forwarding to the output port. Therefore, messages that are crossing a switch suffer from two types of delay. The first type occurs due to the hardware of switch, known as *fabric delay*, which varies in different manufacturers. The second type is because of store-and-forward transmission of messages. The fabric delay is denoted by Δ, while SFD identifies the store-and-forward delay. The switch manufacturer usually gives the fabric delay. Moreover, the store-and-forward delay equals the transmission time of the message.

4.2.2 Traffic model

We consider a set Γ of N messages characterized as follows:

$$\Gamma = \{m_i(C_i, T_i, P_i, D_i, M_{max_i}), i = 1..N\} \tag{4.1}$$

Above, C_i represents the transmission time of the message, T_i denotes the period (or minimum inter-transmission time T_{mit}) and D_i denotes the relative deadline of m_i. The model is deadline constrained, i.e., $D_i \leq T_i$. Moreover, the priority of the message is shown by P_i and M_{max_i} is the maximum packet size among the packets that compose m_i. This means that large messages are fragmented into smaller messages for transmission.

4.2.3 Response time analysis

We present an analysis when we use a two levels hierarchy, i.e., the asynchronous window at the EC level and the flat servers at the asynchronous window level, each associated to one asynchronous message [37].
A message transmitting from its source node to its destination node may suffer from different interference. These interferences include the uplink interference, the downlink interference, switching delay and the effect of idle time in the transmission windows.

4.2.3.1 An illustrative example

To identify the interference messages might suffer and study its impact on response times, we will consider the simple example shown in Figure 4.4 with five nodes connected to one switch and exchanging four unicast messages. We generate this example by simulating the master scheduler within FTT-SE.
The system configuration is the following: duration of EC $LEC = 2000\mu s$, duration of asynchronous window $LW = 819\mu s$, and the maximum packet size for any message m_i, M_{max_i} takes $128\mu s$. Let us focus on message $m_{33}(C_{33} = 1424\mu s, D_{33} = T_{33} = 21EC)$ sent from node A to B, thus comprising 11 MTU-sized packets plus a last packet of $16\mu s$. Since $C_{33} > LW$, at least $2EC$ are required to transmit the message.

The remaining traffic highlights three different types of interference, *uplink* , *downlink* and *indirect*. The former is illustrated by the higher priority message $m_{99}(C_{99} = 1536\mu s, D_{99} = T_{99} = 19EC)$ that shares the same path. In this case, m_{99} interferes with m_{33} both in the uplink and

downlink, and the worst case occurs when they are both ready in the same EC. However, the interference occurs in either uplink or downlink, only, and we need to consider the longest. In this case, the resulting response time is $RT_{33} = 4EC$.

Downlink interference is shown with message $m_{11}(C_{11} = 949\mu s, D_{11} = T_{11} = 17EC)$ sent from node D to B, thus sharing the downlink of m_{33}. The combined interference of m_{11} and m_{99} on m_{33} can now lead to $RT_{33} = 6EC$.

Finally, we add message $m_{51}(C_{51} = 603\mu s, D_{51} = T_{51} = 13EC)$ sent from node D to C to illustrate indirect interference. This message interferes in the uplink with m_{11} that, in turn, interferes in the downlink with m_{33}. In a single switch case, this indirect interference does not further increase RT_{33} that remains 6 EC.

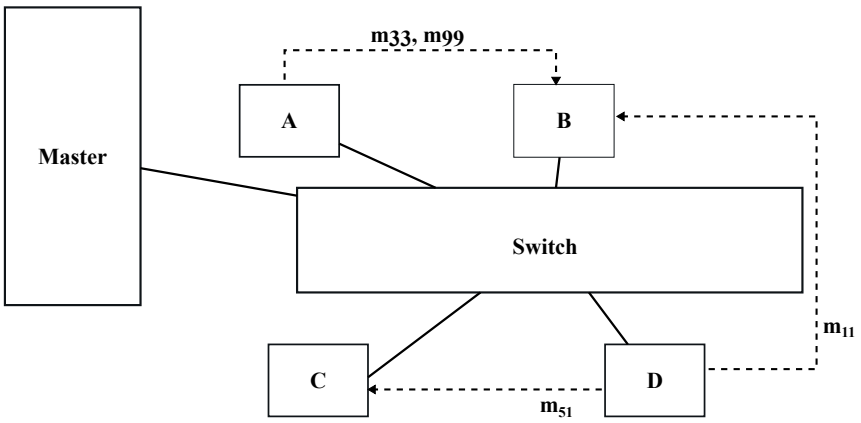


Figure 4.4: Illustrating the sources of interference.

In the following, we define different kinds of interference and sources of delay. Also, we provide the analytic model for computing message response times.

4.2.3.2 Effect of the idle time

The scheduling model of FTT-SE allows us to transmit the scheduled messages within their associated transmission windows. In case the scheduler cannot fit a message inside a window, the transmission of the message is postponed to the upcoming ECs (Figure 4.5). Therefore, the transmission window is not fully used, being partially wasted. The wasted part of the window where no transmission is occurring is called *idle time*. The idle time, in the worst case, is the maximum packet size among the set of messages, which is calculated in (4.2).

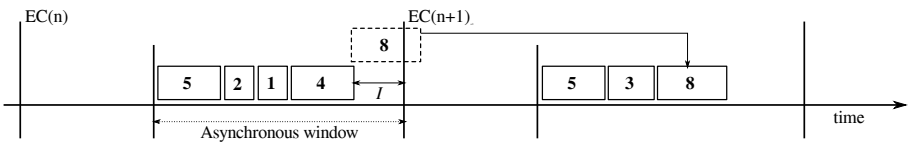


Figure 4.5: Impact of inserted idle time.

$$I \leq \max_{\forall m_j \in \Gamma} \{M_{max_j}\} \tag{4.2}$$

The inserted idle-time does not occur within the flat servers since their capacity is depleted every invocation. However, it may occur once per EC in the asynchronous window. To account for this effect, we define the inflation factor in (4.3), where *LW* is the length of the asynchronous window (which we will also designate by *transmission window*) and *LEC* is the length of the EC. This factor is used to virtually increase the messages transmission times so that, for scheduling purposes, they always cover the full window whenever there is inserted idle-time [138].

$$\alpha = \frac{(LW - I)}{LEC} \tag{4.3}$$

4.2.3.3 Uplink interference

The message under analysis may suffer from higher priority messages transmitting together with the message from the same source node. Therefore, the higher priority interference in the source node should be added into the analysis. The interference is computed in (4.4), where $ul(m_i)$ is the set of messages that share uplink with m_i, $hp(m_i)$ is the set of higher priority messages than m_i and x is the estimated response time.
Also note that, within the ready queue, messages are sorted first according to their priorities, and next by their ids. Messages with higher id take precedence over messages with lower id when they have the same period. This sorting is considered in $hp(m_i)$.

$$Is_i = \sum_{\forall m_j \in hp(m_i) \wedge m_j \in ul(m_i)} \left\lceil \frac{x}{T_j} \right\rceil \frac{C_j}{\alpha} \tag{4.4}$$

4.2.3.4 Downlink interference

Similarly to the uplinks, the message may suffer from higher or equal priority messages in the downlink. This interference is computed in (4.5), where $dl(m_i)$ is the set of messages that share downlinks with m_i.

$$Id_i = \sum_{\forall m_j \in hp(m_i) \wedge m_j \in dl(m_i)} \left\lceil \frac{x}{T_j} \right\rceil \frac{C_j}{\alpha} \tag{4.5}$$

4.2.3.5 Indirect interference

By definition, indirect interference is caused by the messages that delay the messages which create direct interference with the message under study. Following several observations with the simulator, we conjecture that this indirect interfering does not increase the worst-case response time of a message and may even reduce it if sufficiently delaying the message that interferes directly in the downlink. In a single switch network, indirect interference simply affects the moment at which the messages that interfere directly in the downlink arrive at the downlink. In the worst-case, they all

arrive at the downlink at the same time, creating the maximum direct downlink interference that was explained before. Thus, in single switch systems, we can discard indirect interference. This was verified by simulation.

4.2.3.6 Switching delay

As mentioned in the system model, the message crossing a switch is delayed by two parameters, i.e., the fabric delay Δ and the store-and-forward delay. The switching delay is computed in (4.6), where it is also inflated by α. Moreover, note that $SFD_i = C_i$.

$$SD_i = \frac{SFD_i + \Delta}{\alpha} \tag{4.6}$$

4.2.3.7 Final response time

To find the final response time RT_i of m_i, it is required to solve Eq. 4.7.

$$RT_i = x = \frac{C_i}{\alpha} + SD_i + Is_i(x) + Id_i(x) \tag{4.7}$$

Eq. 4.7 can be solved using fixed-point iteration method: $RT_i = min(x^l) : x^l = x^{l-1}$. Iteration starts with $x^0 = \frac{C_i}{\alpha}$. Since deadlines are also expressed as integer numbers of ECs, we round off the analysis estimate to multiples of the EC; analysis reports a deadline violation if $\left\lceil \frac{x^l}{LEC} \right\rceil > D_i$.

4.3 Evaluation

In this section, we describe the system setup used for the experiments, and we also describe the generation of message sets. Finally, we present the experimental methodology and discuss the results.

4.3.1 System setup

The system consists of 10 slave stations connected to a single switch. A station can generate a maximum of 5 messages. The duration of EC; *LEC* is 2000 μs whereas the size of the MTU (Maximum Transmission Unit) is 128 μs. The size of the transmission window *LW* within an EC varies across different experiments.

4.3.1.1 Utilization per link

We consider that each link is scheduled independently using the Rate-Monotonic (RM) scheduling policy [103]. Firstly, we calculate the RM utilization bound for a link $U^{RM} = n(2^{1/n} - 1)$ and reduce it by the factor α to account for inserted-idle time. Thus, maximum available bandwidth for any link is given by $U^{max} = U^{RM} \times \alpha$ [137]. We further reduce this bandwidth to avoid conditions near schedulability threshold. The utilization value thus obtained is the effective bandwidth

available on the link and is denoted by U. Respecting this threshold when generating load per link virtually eliminates non-schedulable sets. Several experiments will use different values of U.

4.3.1.2 Generating a random data set

For each node, the utilization U is distributed among messages being produced at that node using the UUniFast algorithm [139]. We present the pseudo-code of the algorithm that generates a random message set (Algorithm 1). This algorithm has four inputs namely *LEC* the duration of the elementary cycle, U the available bandwidth, n the number of nodes/slave stations, and m the maximum number of messages per node. We select a source node beginning at 1 (line 17), and generate messages being produced therein while choosing a random destination for each message (lines 17 - 20). The choice of different message period (random, harmonic, primes) can be specified. The size of message j is computed using its bandwidth share $\mathbf{V}[j]$ from the utilization vector $\mathbf{V}$ (line 23). This way the uplink utilization is constrained to U by construction. The bandwidth available on a given downlink c is denoted by $\mathbf{U}_{dl}^{c}$. Accounting for the jitter that messages have when they arrive in the downlink, the used bandwidth in downlinks is smaller than U. We use a heuristic where maximum downlink bandwidth is constrained to 70% of U (line 26). A new utilization vector is generated every time we produce messages for a new source node (line 11). The basic operation of the algorithm is thus to distribute the available bandwidth among messages while respecting constraints. The number of messages in a given message set may be smaller than $m \times n$ due to certain non-conforming conditions such as a message smaller than an MTU or when a downlink is full. This procedure generates one random set. To generate a large data set consisting of N message sets, the algorithm is invoked N times.
When adding messages to the set, certain utilization thresholds can be overcome. In that case, we discard the messages. The uplink utilisation is constrained to U, given by the vector $\mathbf{V}$ and distributed across m shares. The producer station starts as station 1 and then incremented by 1 until reaching station n (equal to the number of available stations). Each message will use the bandwidth share from the utilisation vector $\mathbf{V}$ which is limited to m places. However, for each message, both source and destination must be chosen; thus, we add the same utilisation to the downlink, chosen each time randomly among the n stations except the producing one. Downlink utilization is constrained to 70% U accounting for the jitter conditions[1].When a message does not fit in a given link, it is discarded (removed from uplink and downlink).

4.3.2 Experiments

In the following, we describe the experiments we carried out with many random message sets. For a given message set, we execute an implementation of the FTT-SE master scheduler as well as run

[1]The 70% U bound on $\mathbf{U}_{dl}^{c}$ is not firm. In fact, as shown in lines 26- 27 of Algorithm 1, messages whose individual utilization causes crossing that threshold are still accepted. Note that schedulability will be tested afterwords, anyway, thus this is safe. The 70% U constraint aims, merely at facilitating the generation of schedulable sets since the arrival of messages at the downlinks will be normally jittered, increasing the chances of non-schedulability

Algorithm 1 Generating a random message set

Input: LEC, U, n, m

1: **procedure** Glds ▷ Generate Large Data Set
2: $\mathbf{V} \leftarrow$ **UUniFast**(m, U)
3: $\mathbb{A} \leftarrow 1 : n$
4: $j \leftarrow 0$
5: $i \leftarrow 1$
6: msg_id $\leftarrow 0$
7: $p \leftarrow 1$
8: **while** $i \leq m \times n$ **do**
9: $j \leftarrow j + 1$
10: **if** $j = m + 1$ **then**
11: $\mathbf{V} \leftarrow$ **UUniFast**(m, U)
12: $p \leftarrow p + 1$
13: **if** $p > n$ **then** ▷ limit the number of producers to n
14: **return**
15: $j \leftarrow 1$
16: msg_id $\leftarrow$ msg_id $+ 1$
17: msg_producer $\leftarrow p$
18: $\mathbb{S} \leftarrow \mathbb{A} \setminus p$
19: $c \leftarrow$ **randi**$(\mathbb{S})$ ▷ choose a random consumer, different from the source
20: msg_consumer $\leftarrow c$
21: msg_period $\leftarrow$ **randi**$(5, 70)$
22: msg_deadline $\leftarrow$ msg_period
23: msg_size $\leftarrow \mathbf{V}[j] \times$ (msg_period $\times LEC$)
24: **if** msg_size $<$ MTU **then**
25: **continue**
26: **if** $\mathbf{U}^c_{dl} \leq 0.70 \times U$ **then**
27: $\mathbf{U}^c_{dl} \leftarrow \mathbf{U}^c_{dl} + \mathbf{V}[j]$
28: **else**
29: **continue**
30: $i \leftarrow i + 1$

the analysis program. We then compare the longest observed message response times (RT_o) with the respective analytic upper bounds (RT) (Figure 4.6).

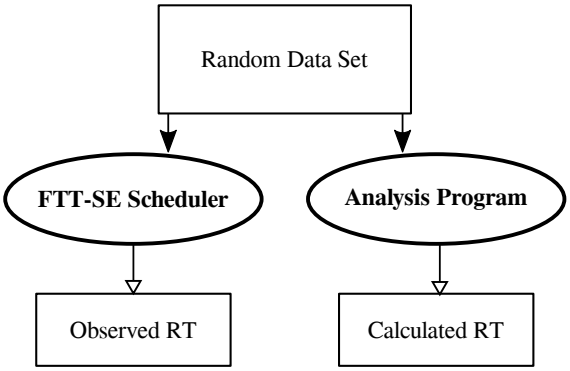


Figure 4.6: Experimental method

4.3.2.1 A single set

First, we show the results for a single data set with a trace of 15000 ECs. We plot a message's calculated and observed response times as well as its period in number of elementary cycles (Figure 4.7). We see these values along each vertical line for any message in the set. The figure shows that the deadlines, considered equal to their periods, are all met during the period of observation and similarly that all response time values, observed and computed, are shorter than the corresponding periods. We can also see some cases with an exact match between the observed and calculated response time values, which indicate accurate analytical estimates.

4.3.2.2 Multiple simulations

We are interested in quantifying the analysis pessimism when system configuration changes, such as properties of the message set, network configuration parameters or protocol configuration parameters. To address this concern, we carry out multiple simulations over several thousands of message sets. In these simulations, we log results extracted from each data set and then compute values of interest over the complete simulation run. In particular, we compute the following values: *percent matches*, this value gives the percentage of cases in the dataset in which observed response time values are equal to the calculated values; *difference above maximum*, in a data set, we choose the message which calculated and observed response time differs the most, and we calculate its percent increase of the calculated over the observed values $((RT - RT_o)/RT_o) \times 100\%$. Finally, trying to better represent the pessimism across the whole data set, we calculate the percent increase for all the messages referred to as *percent increase of analysis over observed.*

4.3.2.3 Impact of activation pattern

In this experiment, we study the impact of message activation pattern on schedulability. For this case, we generate 1200 message sets with their minimum inter-arrival periods (given in number

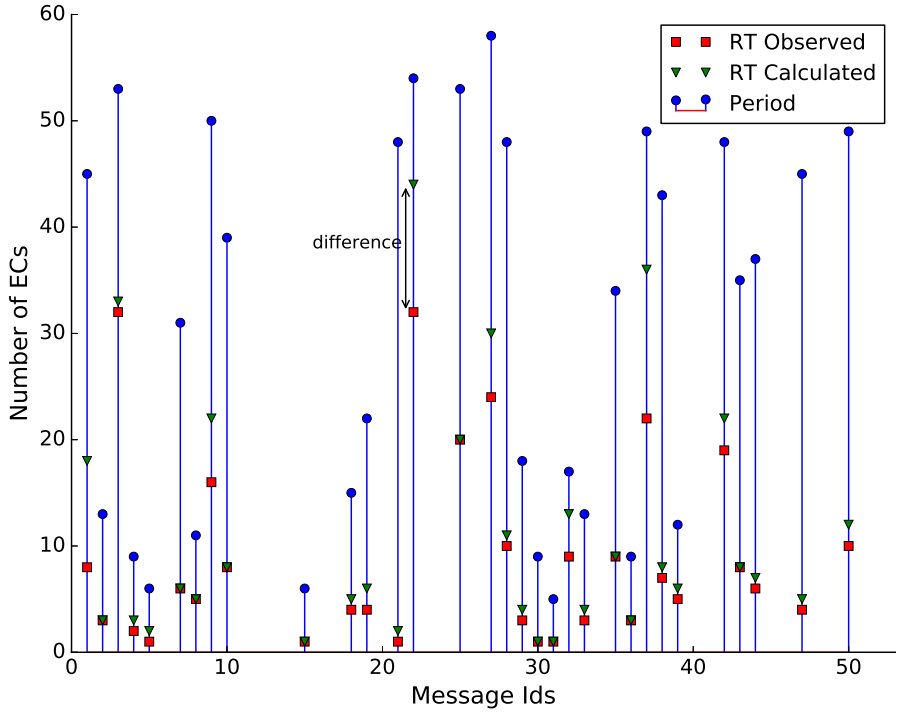


Figure 4.7: Response times of messages in a random set as observed with implementation and calculated with analysis program

of ECs) taken randomly from the set $\{4, 8, 16, 32, 64, 128\}$ and we trigger each message set periodically (with no offsets) as well as sporadically. With sporadic activation, m_i's next activation is triggered at an *EC* randomly chosen in the interval $(T_i, 2T_i - 1)$ from its previous activation. We consider RM priority assignment and we break ties by giving higher priority to higher Id.

In each case, FTT-SE master scheduler runs 20000 elementary cycles. We measure message maximum response time (RT_o) from this simulation. Then, for each message, we take the difference between RT_o values reported with its periodic and sporadic activation. For the message set, we count the number of messages that reported larger RT_o with each activation pattern, and the number of messages that experience the same RT_o. Our observations show that no activation pattern favours the other significantly. Results indicate that, on average, the three cases are evenly distributed each representing 33% of the total cases (Figure 4.8).

Figure 4.8 shows the histograms of the percentage of messages per set that meet the condition of each case, namely larger periodic RT_o, larger sporadic RT_o, periodic RT_o equals sporadic RT_o. In blue we can see how many sets have a certain percentage of messages with larger periodic RT_o. There is a predominance of sets with 10% to 30% messages reporting larger periodic RT_o but the distribution shows a long tail to the right, i.e., higher percentages, up to a few sets with 100% messages exhibiting larger periodic RT_o. On the other hand, the orange histogram reveals that the majority of the sets have between 25% and 50% of messages with larger sporadic RT_o. The distribution of the percentage of messages per set that exhibit equal periodic and sporadic RT_o is shown in yellow and we also see a concentration between 30% and 50% messages. In this case,

though, the distribution extends more to the left, i.e., lower percentages, down to several sets with no messages reporting equal periodic and sporadic RT_o.

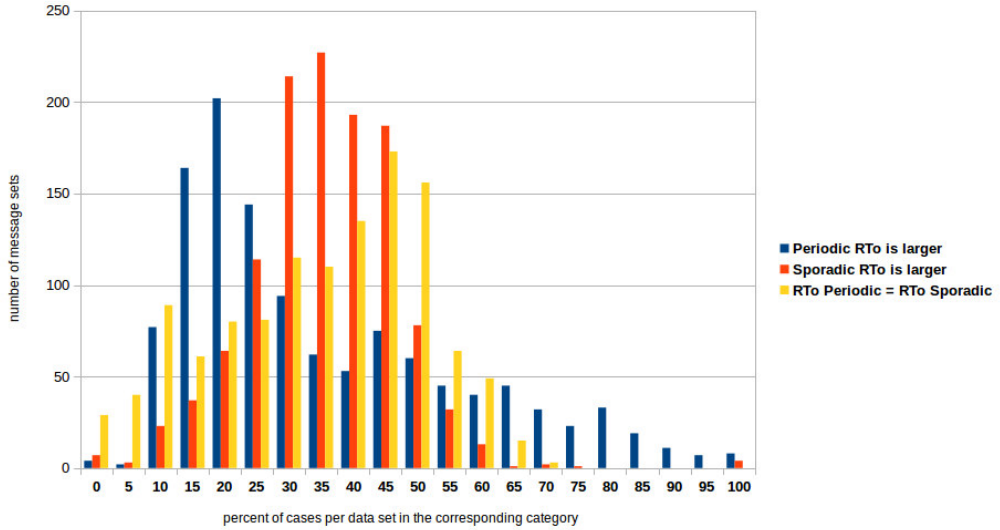


Figure 4.8: Histogram of the percentage of messages per data set for each category

Interestingly, we would expect RT_o for periodic activations to be larger given their higher load. Note that the average inter-arrival periods of the sporadic case are $1.5 * T_i$. Moreover, releasing the periodic messages simultaneously at $EC = 0$ leads to a well defined worst-case busy interval at least in the uplinks, which is easy to capture in the observations. Yet, a significant number of sets with sporadic activations present RT_o larger than or equal to that of the periodic case. In our system, however, with multiple links, simultaneous message transmissions do not interfere if they do not share links. This effect softens the difference between both cases.
To illustrate one case in which a given message triggered sporadically exhibits a larger RT_o than when triggered periodically, we provide the following example. We choose message m_{43} from one data set that reports $RT_o = 14$ with periodic activation and $RT_o = 31$ when messages in the set are activated sporadically. Table 4.1 lists the messages that share links with m_{43}.

We analysed the FTT-SE scheduler simulation traces to observe the ready queue state during the time between activation and dispatch of m_{43} in each case. Figure 4.9 shows these traces displaying the messages in Table 4.1, only, for the sake of clarity, and highlighting new message activations. We observe m_{43} worst-case response time that occurred after activation in EC_{4107} and dispatch in EC_{4138}. For the periodic case, RT_o occurs after activation in EC_0 and dispatch in EC_{14}. This is when all messages are released simultaneously in EC_0. However, the worst-case pattern is a complex combination of interferences in the uplinks and downlinks. In this case, the high initial backlog triggered by the periodic release generates the maximum interference in the uplinks but not in the downlinks. Conversely, sporadic releases, given their random nature, tend to be more efficient in finding pernicious interference patterns in the downlinks.

Moreover, note that messages are composed of multiple packets and might need multiple ECs to complete transmission. In the referred example, m_{41} is activated simultaneously with m_{42} in

Table 4.1: Interference set of message m_{43}

Description	Message(s)	$T_i(EC)$	Src_i	$Dest_i$
Message under study	m_{43}	128	9	2
Uplink interference	m_{41}	128	9	6
	m_{42}	32	9	8
	m_{45}	16	9	8
Downlink interference	m_{25}	8	5	2
	m_{19}	128	4	2
	m_{28}	4	6	2
	m_{35}	64	7	2
	m_{37}	16	8	2
	m_{48}	128	10	2
	m_{50}	8	10	2

EC_{4105} (not shown in the figure). Priority order is maintained and m_{42} is scheduled before m_{41}. m_{42} is a large message (35 fragments). m_{42} is only partially scheduled by EC_{4107} in which m_{43} arrives. So, by virtue of sporadic activations, there is a sort of priority inversion. m_{41} has a lower priority but m_{43} must wait for it since it precedes m_{43} in the queue. However, if m_{41} is blocked due to interference on its downlink, m_{43} can be scheduled.

4.3.2.4 Harmonic vs Primes

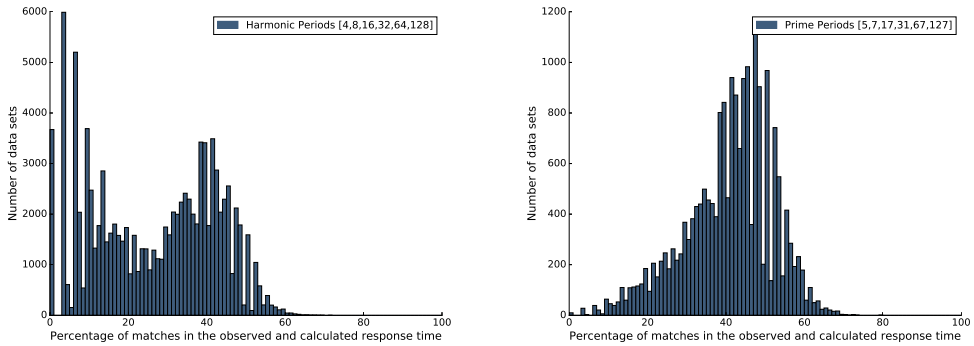
In this experiment, we try to see how does the choice of certain period values impacts on the analysis efficiency. We compare harmonic and prime periods. Harmonic periods are chosen from the set $\{4,8,16,32,64,128\}$ and prime periods are chosen from the set $\{5,7,17,31,67,127\}$. EC duration $LEC = 2000\mu$s and asynchronous window size $LW = 1049\mu$s. The available utilization U to generate the data set is approximately 31%. To exhibit all the interference patterns, FTT-master scheduler should execute at least until the LCM of message periods. This is possible with harmonic periods, but for the case of prime periods ($LCM = 156948505$), it is prohibitive in terms of memory and time, in particular when we have to simulate over several thousands of data sets. Therefore, our simulation trace for harmonic case consists of 400 ECs (which is $>\sim 3$ LCM), and for the case of primes it is 15000 ECs. Also, the data set size is 99936 and 19991 message sets respectively for harmonic and prime cases.

We can observe in Figure 4.10 that distribution for prime periods is more concentrated towards higher matches whereas there are fewer matches with harmonic periods. The average number of matches for message sets with prime periods is 40% of the total messages in the set, whereas, for harmonic sets, the average is at 25%.

We observe in Figure 4.11 that for both cases i.e., primes and harmonic, the majority of the datasets report analytic upper bounds that are up to 6 times the observations. However, the analysis of primes fares better than that of harmonic; we further calculate that with prime periods, analysis

EC_{4107}: m_{25} – m_{42} – m_{41} – m_{43}	EC_0: m_{28} – m_{50} – m_{25} – m_{45} – m_{37} – m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4108}: m_{42} – m_{41} – m_{43}	EC_1: m_{25} – m_{45} – m_{37} – m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4109}: m_{50} – m_{42} – m_{35} – m_{41} – m_{43}	EC_2: m_{45} – m_{37} – m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4110}: m_{35} – m_{41} – m_{43}	EC_3: m_{45} – m_{37} – m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4111}: m_{28} – m_{35} – m_{41} – m_{43}	EC_4: m_{28} – m_{45} – m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4112}: m_{35} – m_{41} – m_{43}	EC_5: m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4113}: m_{35} – m_{41} – m_{43}	EC_6: m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4114}: m_{35} – m_{41} – m_{43}	EC_7: m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4115}: m_{45} – m_{41} – m_{43}	EC_8: m_{28} – m_{50} – m_{25} – m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4116}: m_{25} – m_{45} – m_{41} – m_{43}	EC_9: m_{25} – m_{42} – m_{35} – m_{43} – m_{41} – m_{19}
EC_{4117}: m_{28} – m_{45} – m_{41} – m_{43}	EC_{10}: m_{43} – m_{41} – m_{19}
EC_{4118}: m_{45} – m_{41} – m_{43}	EC_{11}: m_{43} – m_{41} – m_{19}
EC_{4119}: m_{45} – m_{41} – m_{43}	EC_{12}: m_{28} – m_{43} – m_{41} – m_{19}
EC_{4120}: m_{41} – m_{43}	EC_{13}: m_{43} – m_{41} – m_{19}
EC_{4121}: m_{41} – m_{43}	
EC_{4122}: m_{41} – m_{43}	
EC_{4123}: m_{50} – m_{41} – m_{43}	
EC_{4124}: m_{28} – m_{41} – m_{43}	
EC_{4125}: m_{37} – m_{41} – m_{43}	
EC_{4126}: m_{37} – m_{41} – m_{43}	
EC_{4127}: m_{25} – m_{41} – m_{43}	
EC_{4128}: m_{28} – m_{41} – m_{43}	
EC_{4129}: m_{41} – m_{43}	
EC_{4130}: m_{41} – m_{43}	
EC_{4131}: m_{41} – m_{43}	
EC_{4132}: m_{45} – m_{43}	
EC_{4133}: m_{45} – m_{43}	
EC_{4134}: m_{28} – m_{50} – m_{45} – m_{43}	
EC_{4135}: m_{45} – m_{43}	
EC_{4136}: m_{25} – m_{45} – m_{43}	
EC_{4137}: m_{43}	

Figure 4.9: Reduced simulation trace for message m_{43}: sporadic (left) versus periodic release (right)

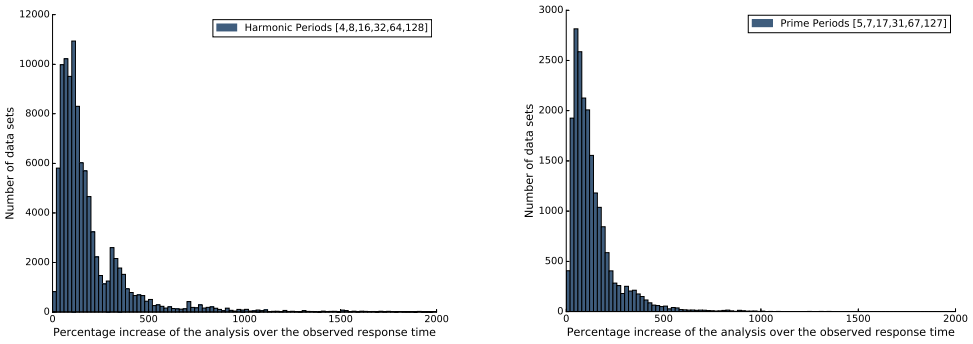


(a) Harmonic (b) Primes

Figure 4.10: Percentage of matches between calculated results and observed values

exceeds the observations by 6 times (500%) or larger for under 2% of the total datasets. With harmonic periods, however, there are approximately 6% such data sets.

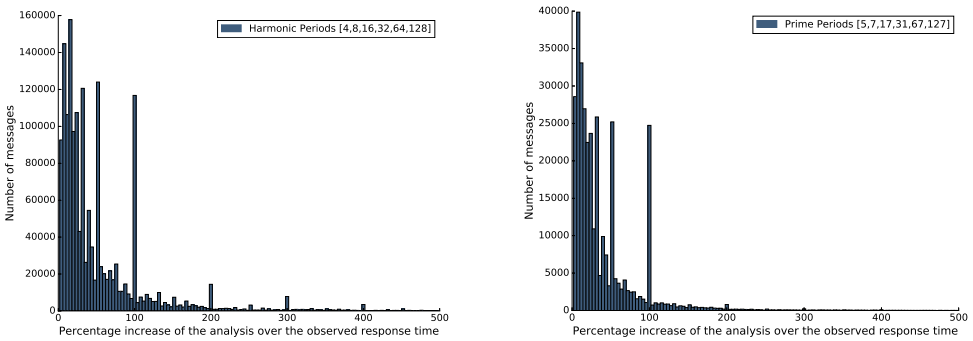
We generate another result that shows percent increase of analysis over observed values but for



(a) Harmonic (b) Primes

Figure 4.11: Percentage increase in RT with analysis method over observed values

all the messages in the data set (Figure 4.12). Since there are on average 25% and 40% matches between RT_o and RT values per data set (see Figure 4.10), hence this result will have a long peak at the point 0 on the horizontal axis indicating the cases of an exact match. This, however, makes the plot readability poor, therefore, we start the plot with percent increase at 2% and keep the horizontal limit to 500% (6 times increase). This range already covers the majority of data points. There are about 0.66% of messages with harmonic periods that report 6 times or larger increase (500% increase) whereas with prime periods this number is only about 0.08% of total messages.



(a) Harmonic (b) Primes

Figure 4.12: Percentage increase in RT with analysis method over observed values, displaying percentages above 2%, only, to omit the peak at 0%

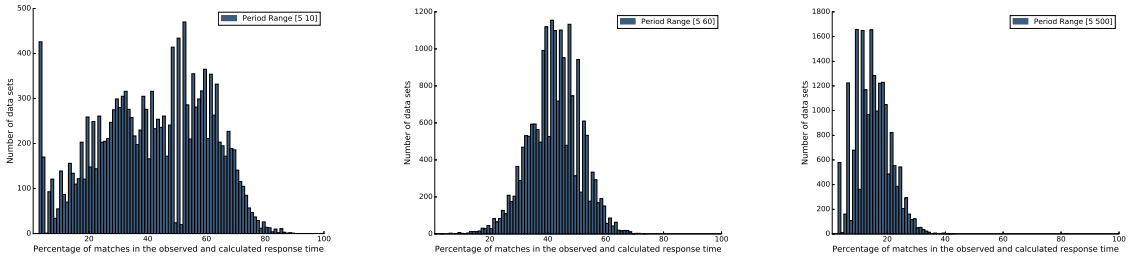
4.3.2.5 Different period range

Next, we experiment using different range of periods for the message sets. We choose message periods within three ranges, namely *short* $[5\cdots10]$,*medium* $[5\cdots60]$ and *long* $[5\cdots500]$. The size of the asynchronous window is $LW = 1049\mu$s. The available utilization U to generate datasets is approximately 31%. The simulation trace is 15000 ECs long. The dataset size is 19161 message sets for short period range and 20000 for other ranges. Figure 4.13 shows more matches with short period range, with decreasing matches for the large period range. On average there are 33%,

41% and 14% matches between analysis and observations. On average, short period range reports smaller matches than medium range, but there is a larger concentration with higher matches per set; using short periods, 29% of the message sets report matches in over 50% of the messages in the respective sets whereas only 15% of the message sets with medium period range report such match.

In Figure 4.14, we can see the maximum percentage increase over observed response time reported with analysis program (i.e., *diff_above_max*). The calculations show that for periods in short range, there are no datasets with an increase in RT of 6 times or greater, whereas for the medium and long range of periods these values are 0.085% and 0.21% respectively.

In Figure 4.15, we see the percentual increase of the analytical over observed response times computed for each message in the set. In particular, the number of messages that report an increase of 3 times or larger is only at 0.06%, 0.16% and 0.68% of messages, respectively from experiments with the short, medium and long range of periods. An immediate observation is that, in general, analytical upper bounds do not exceed the observed values significantly. Since we keep the link utilization fixed for the whole data set, message sizes can be very large with several fragments when periods are long and smaller with short periods. Also, since the data sets are generated following a sanity check on their schedulability, there is a higher probability for matches when the range is short. For long periods, the response times RT and RT_o, as well as their absolute difference, can be larger compared with the case of short periods, but percentually smaller. This explains the results shown in Figure 4.15.

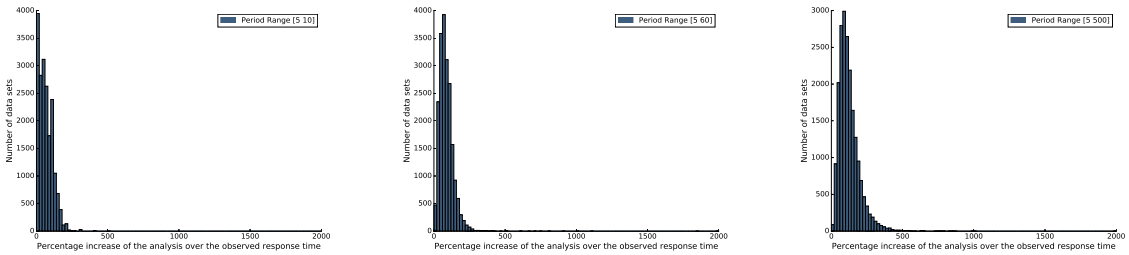


(a) Period $= [5 \cdots 10]$ (b) Period $= [5 \cdots 60]$ (c) Period $= [5 \cdots 500]$

Figure 4.13: Percentage of matches between RT and RT_o for different period ranges

4.3.2.6 Different window size

In this experiment, we change the size of the asynchronous window LW and observe its impact on the analysis efficiency. We keep the same data set across different experiments. This allows us to observe the variation in message response times as the scheduling window is enlarged. We report experiments with three different window sizes (in microseconds) being $\{524, 874, 1574\}$. The chosen period set is harmonic $\{4, 8, 16, 32, 64, 128\}$, which allows us to execute the FTT-SE scheduler for only 400 cycles and hence experiment with a large dataset i.e., 98795 message sets while keeping the simulation time small.



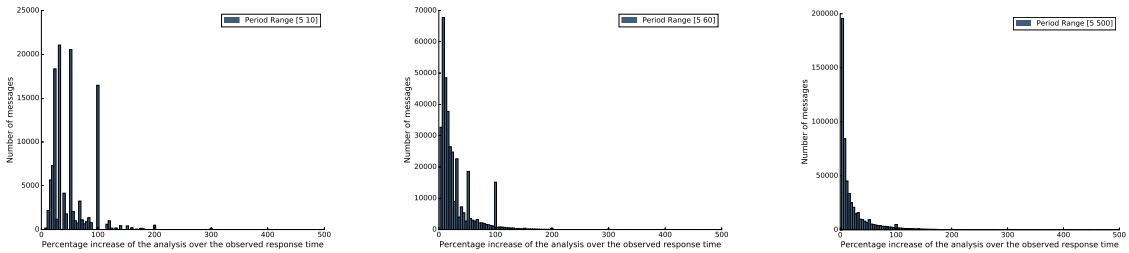
(a) Period $= [5 \cdots 10]$ (b) Period $= [5 \cdots 60]$ (c) Period $= [5 \cdots 500]$

Figure 4.14: Maximum percentage increase of RT over RT_o in the data set, varying the period range

When varying network configuration parameters, we use harmonic periods. Despite having observed a lower efficiency of the analysis in these conditions, we believe that using a message set with prime periods would lead to similar effect in face of variations of the network configuration. The approximate uplink and the downlink utilization values are respectively, 14% and 10%. Figure 4.16 shows the results of matches between the analysis and observations. We can see that as the window is enlarged from 524μs to 1574μs, the frequency of matches between the analysis and observation increases. On average there is 9%, 36% and 60% matches in the three cases. We observe in Figure 4.17 the maximum percentage increase over observed response time reported with analysis program (i.e., *diff_above_max*). The number of datasets where analytic upper bounds exceeded the maximum observed response times by 6 times or greater is 10%, 5.62%, and 0.118% of total datasets respectively for LW size 524μs, 874μs and 1574μs. In Figure 4.18 we see the percent increase of the analytical RT over observed values for all messages in each dataset. For better readability, the plot shows percent increase up to 500% or 6 times. This range already covers the majority of data points. There are 5.45%, 1.41% and 0.14% of total messages respectively in experiment with 524μs, 874μs and 1574μs LW that report 3 times or larger increase in RT. A change in LW has an impact on the inflation factor α and hence on the response time estimation (RT). In general, larger LW increases α. Since $\alpha < 1$, and is applied as a divisor, for the same message the inflated C decreases as LW and hence α increases, directly reducing final RT (see eq. 4.4 and 4.5). Conversely, a decrease of RT_o is intuitive since all messages are multi fragments, hence, there is a larger resource available to schedule more packets with larger LW. Since both RT and RT_o reduce, so the difference becomes smaller and smaller. This explains the smaller differences and more matches as LW increases.

4.3.2.7 Different link utilization

In this experiment, we change the link utilization and observe its impact on the analysis. We experiment with 20%, 60% and 90% of the guaranteed schedulable capacity of the asynchronous window, which corresponds to a link utilization of $U^{RM} \times \alpha$. Other configurations include, $LEC =$ 2000μs and $LW = 1049\mu$s. Message periods are harmonic chosen randomly from the set $\{4, 8, 16, 32, 64, 128\}$ allowing a short simulation trace of 400 ECs. The dataset size is 100000, 99710 and 99936 sets



(a) Period $= [5 \cdots 10]$ (b) Period $= [5 \cdots 60]$ (c) Period $= [5 \cdots 500]$

Figure 4.15: Percentage increase of RT over RT_o computed over all the messages in the data set, varying the period range

for the 20%, 60% and 90% cases, respectively. Table 4.2 shows the corresponding uplink and downlink utilizations of the three cases.

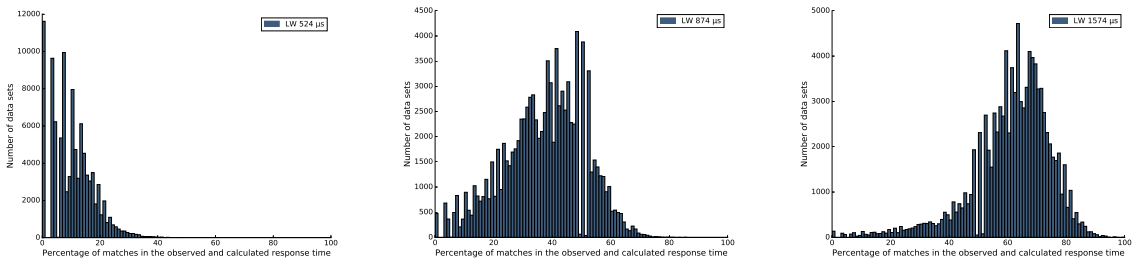
Table 4.2: Experiment utilization values

U^t	$U^{max} = U^{RM}(\alpha)$	$U_{ul} = U^t \times U^{max}$	$U_{dl} = 0.70(U_{ul})$
0.2	0.34077	0.068	0.0476
0.6	0.34077	0.2044	0.1431
0.9	0.34077	0.3066	0.2146

Figure 4.19 shows that the number of matches between RT_o and RT decreases as we increase the link utilization. The average matches per message set are 58%, 29% and 19% for the three link utilization cases, respectively.
In Figure 4.20, we see the maximum percentage increase over observed response time reported with analysis program (i.e., *diff_above_max*). The calculations show that for experiments with the three cases of growing link utilization there are respectively 1.53%, 4.47% and 6.073% of data sets that report a maximum increase of 6 times or larger.
Figure 4.21 shows the percentual increase of RT over RT_o for each message in the whole data set. The number of messages with RT greater than 3 times RT_o make approximately 0.5%, 1.43% and 3.37% of the total number of messages in the experiment for the three link utilization, respectively. Concerning the 20% case, we observe that the size of the generated sets may be smaller. This may happen when the initial U provided to Alogrithm 1 is small, leading to a reduction in the interference within the message sets thereby reducing both RT and RT_o. The percentual difference reduces, too, leading to more efficient analysis. On the other hand, sets with larger utilization experience more interference and hence longer response times. The differences are also larger in percentage, meaning that the analytical estimates grow more than the actual observations.

4.4 Lessons Learnt

Schedulability analysis verifies a system's timing using a model of the system and aims to capture the worst case scenario of the system in execution. The efficiency of the analysis depends on how

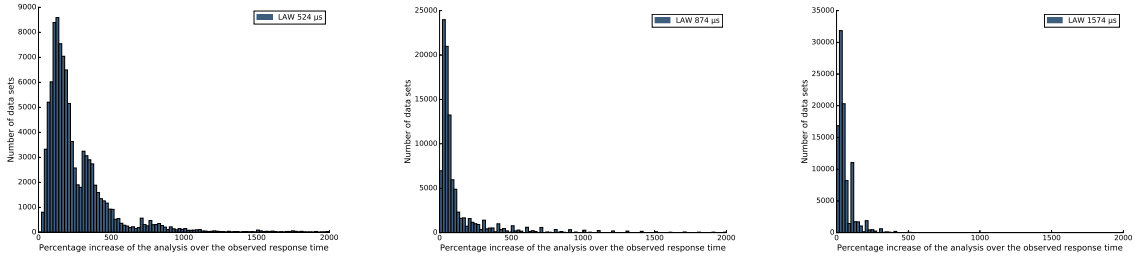


(a) LW $= 524\mu$s (b) LW $= 874\mu$s (c) LW $= 1574\mu$s

Figure 4.16: Percentage of matches between RT and RT_o for different values of LW

well the model can reflect the system characteristics. Given parameters of the system, if the worst case estimates reported with the analysis are within certain bounds (as decided by the designers), the users have confidence that the system will function as desired. Having such a confidence is of particular importance as the system's criticality increases. There are numerous analyses for several protocols which are currently in use such as AFDX, AVB, CAN etc. And, there is a continuing effort to provide further analyses for the same system which are less pessimistic. This effort is carried out so that designers may improve systems' efficiency, for instance, by allowing a higher system utilization. The work in [50] lists several issues facing designers with regard to function partitioning and subsystem integration, which arise in particular due to the real-time or reliability constraints. For such a case, choosing a design which allows more applications, is desirable. To make such a decision at this stage, the study can inform the designer.
In our case, the exercise has led us to make a couple of changes in the analysis to better reflect the protocol, and thereby decrease the pessimism of the analysis estimates. One such change is discounting *indirect interference* (Section 4.2.3.5) from the computation of final response time, which was initially account for. In another change, we reduced the size of interference sets by following a scheduling condition of the FTT-SE protocol that breaks ties among messages with the same priority upon link access.
Moreover, by running simulations using large random data sets, we are able to observe how the efficiency of the analysis varies and configuration where analysis is less accurate.

For example, between harmonic and prime periods, analysis of the primes is closer to the actual worst-case response times that occur at run-time. This is expected since this periods pattern generates rich relative offsets among the messages (generated periodically) that will trigger worst-case conditions both in uplinks and downlinks. Conversely, if the periods are harmonic, the analytical worst-case conditions may never occur, either in uplinks or downlinks, as long as the messages are triggered periodically. This suggests a need for a more accurate analysis in this particular case, when we have asynchronous messages that are essentially triggered periodically with a period equal to their minimum inter-transmission time and their periods are harmonic.
Application period choice depends, mainly, on the underlying application dynamics. For instance, a higher sampling rate in a control loop would imply a smaller period of the sampling task. Traditionally, an effort has been invested to find periods such that the system is schedulable [140, 141].



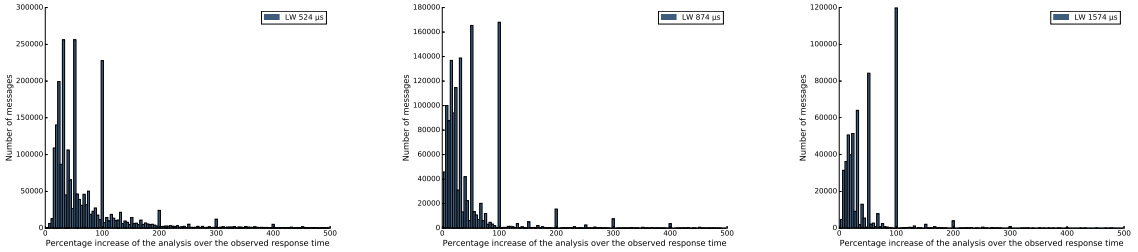
(a) $\mathsf{LW} = 524\mu\mathrm{s}$ (b) $\mathsf{LW} = 874\mu\mathrm{s}$ (c) $\mathsf{LW} = 1574\mu\mathrm{s}$

Figure 4.17: Maximum percentage increase of RT over RT_o in the data set, varying LW

In this work, however, we consider schedulable sets, and we aim to empirically find out in which ranges of periods make the analysis more or less accurate. We must take into account that in our experiments we keep the link utilization fixed for the whole data set. Thus, message sizes vary for different period ranges, being smaller with short periods and can be very large with several fragments when periods are long. We observed that the analysis is less efficient when the period range is longer, involving long periods. However, using long periods can be of interest, thus making it worth improving the analysis for this case. As noted in [141], overload in real-time systems may be dealt with using very long task periods or, in other words, executing jobs less frequently.
When a given work-load is schedulable with different transmission window sizes, the designer can choose larger windows when faster response times are desired. On the other hand, smaller window improves the bandwidth efficiency and leaves room for more applications. However, the analysis is clearly less efficient in this case, thus improvements would be desired for scheduling within very constrained partitions, too.
Finally, when the link utilization of the message set is low, the scheduling is less constrained and the analysis becomes more efficient, too. On the other hand, increasing the link utilization led to a degradation of the analysis efficiency. Unfortunately, increasing the utilization of partitions, as the asynchronous window, in this case, is typically desired thus improving the analysis, in this case, would also be worth.

4.5 Summary

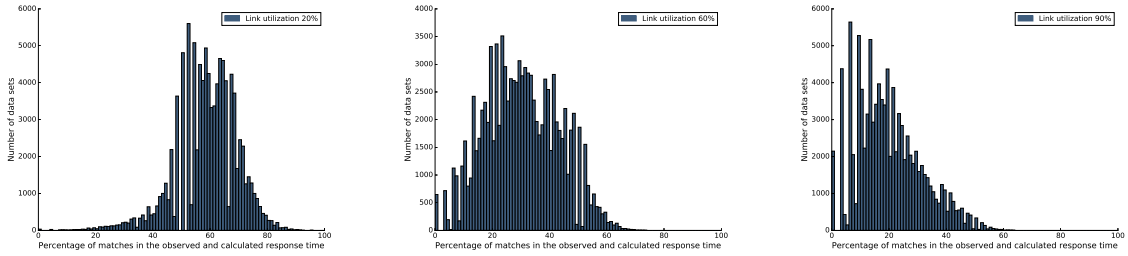
Ethernet is a promising networking candidate for emerging distributed embedded systems. For such systems, reservations provide the means to support composability and address complexity, but they compromise bandwidth efficiency, too. In this chapter, we addressed the network reservations for asynchronous messages within FTT-SE and we presented an extensive simulation to assess the efficiency of a response-time analysis for those reservations. In general, we saw that the analysis shows significant matches with the observations and just a few instances show more than 3 times, sometimes more than 6 times, the observations. We also saw that larger LW reduces the analysis pessimism but at the expense of requiring more resources, and that a short range of periods also improves the analysis accuracy, exhibiting more matches. Nevertheless, we also found



(a) $\mathsf{LW} = 524\mu$s (b) $\mathsf{LW} = 874\mu$s (c) $\mathsf{LW} = 1574\mu$s

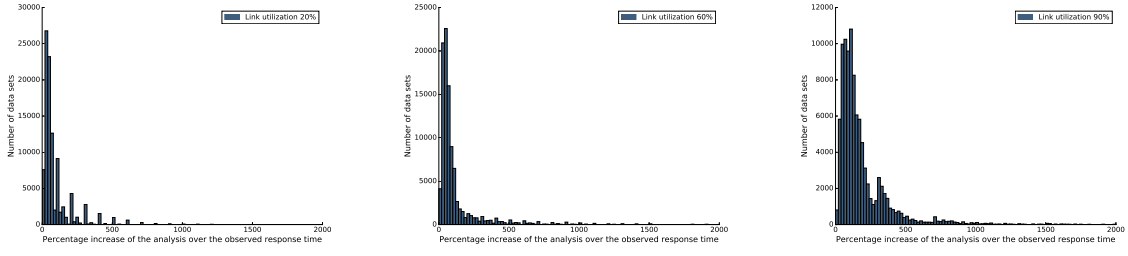
Figure 4.18: Percentage increase of RT over RT_o computed over all the messages in the data set, varying LW

that using longer period ranges reduces the average analysis pessimism. Lastly, in a more expected note, we saw that lower link utilization lead to less pessimistic analysis. This study also identified scenarios where the current analysis can be improved, namely harmonic periods, small partitions and high partition utilization. Improving the analysis is left for future work.



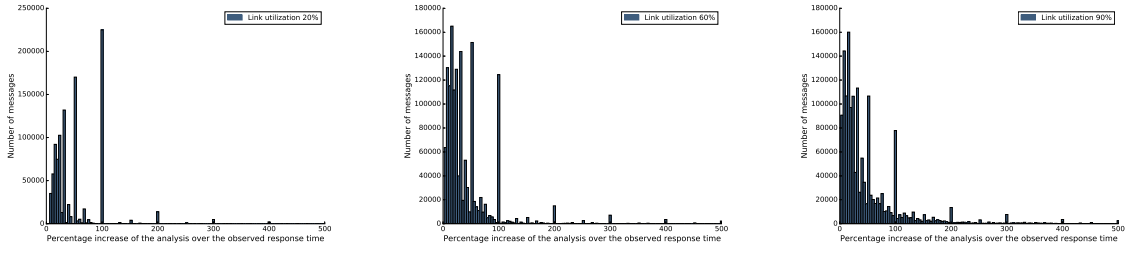
(a) Lower link utilization (b) Mid link utilization (c) High link utilization

Figure 4.19: Percentage of matches between RT and RT_o varying link utilization



(a) Lower link utilization (b) Mid link utilization (c) High link utilization

Figure 4.20: Maximum percentage increase of RT over RT_o in the data set, varying link utilization



(a) Lower link utilization (b) Mid link utilization (c) High link utilization

Figure 4.21: Percentage increase of RT over RT_o for each message in the data set varying link utilization

Chapter 5

Supporting Hierarchical Reservations within FTT-SE using Polling Servers

When multiple applications co-exist in the system, or we have applications with multiple components, flat reservations are not adequate to provide the desired level of isolation between different applications and to meet their timing requirements. For this reason, we use hierarchical reservations; bandwidth is partitioned at multiple levels, and different partitions are assigned to different applications. In this scope, Hierarchical Scheduling Framework (HSF) is instrumental to efficiently deploy hierarchical reservations while also providing run-time temporal isolation between various applications. In this chapter, we show how hierarchical scheduling can be efficiently implemented using Ethernet with ordinary COTS switches and FTT-SE protocol. The scheduling and replenishment of reservations is managed with the Polling Server (PS) policy. This chapter also presents a response-time analysis of the traffic submitted within each partition. Finally, we report some experimental results that validate the analysis and show that different partitions in the network achieve mutual temporal isolation. The contributions of this chapter are the following:

- A multi-level hierarchical server-based scheduling architecture for Ethernet that works on ordinary COTS switches based on the asynchronous communication services of FTT-SE.
- A response-time analysis of the real-time traffic within the servers in any level of the hierarchy.
- A reference implementation that validates the proposed architecture.

This work follows closely the work carried out for the HaRTES switch [129] in which a hierarchical scheduling architecture for Ethernet was also proposed. However, that work relies on a tailored switch that has embedded support for hierarchies of servers in each port while in this work we use FTT-SE, a software-only solution that works on top of ordinary COTS switches. In [128], the authors presented a proof-of-concept implementation of servers within FTT-SE that was called Server-SE. However, this preliminary work missed a full perspective regarding hierarchical scheduling deployment and system analysis. The work reported in this chapter will

complement the work in [128] with an analysis similar to that presented in [129] but considering the differences given the use of FTT-SE and ordinary COTS switches.
The chapter is organised as follows: Section 5.1 presents the concept of hierarchical server-based traffic scheduling and its integration in switched Ethernet. This section also describes the servers scheduling algorithm. Section 5.2 presents a response-time schedulability analysis. In Section 5.3 we present experimental results from a prototype implementation that validate the analysis and show the practicality of the proposed architecture. Finally, we summarise the chapter in Section 5.4.

5.1 Hierarchical Scheduling Framework in FTT-SE

The Hierarchical Server-based Scheduling (HSS) framework is formed by a set of servers connected hierarchically in a tree structure. Each server manages a fraction of the network bandwidth that it will provide to its children servers or streams as shown in Figure 5.1. Associated with each server is a scheduler, a set of child servers or streams and an interface that specifies its resource requirements. The streams are connected to the leaf servers of the tree, and they constitute the actual application load that will consume the network bandwidth. When a server is scheduled, it selects one of its ready children servers. The servers and streams scheduling is carried out by applying an online scheduling algorithm. Then the scheduled child server will also use its scheduler to select another child server, and the same procedure will be repeated down the tree until we reach a leaf server which will finally schedule a message for transmission. The amount of bandwidth given to the scheduled stream is limited by the remaining capacities of all parent servers and the consumed bandwidth by the stream is decreased from the remaining capacity of the leaf server and all the respective parent servers in the tree up to the root server. If the remaining capacity of a server is exhausted, the server becomes suspended until its capacity is replenished.

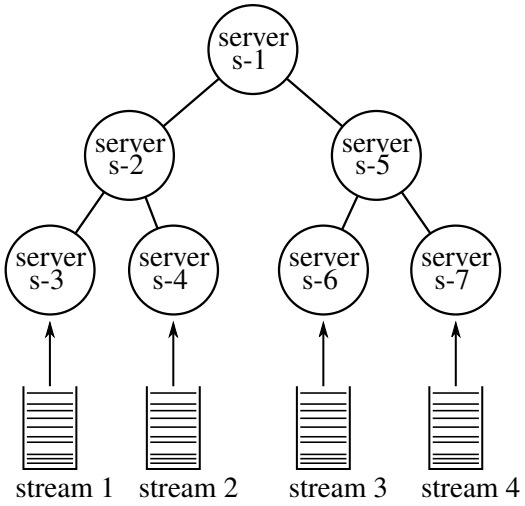


Figure 5.1: An example server hierarchy. Bandwidth is allocated to each server. Application messages arrive at the leaf servers.

5.1.1 Servers integration in FTT-SE: an architectural overview

For reference, an overview of the FTT-SE protocol is presented in Chapter 2, Section 2.9. We consider asynchronous traffic which is scheduled within the asynchronous window of the EC. FTT-SE inherently provides mutual isolation between two different traffic classes, i.e., synchronous and asynchronous by constraining each class to its respective reservation. The parent reservation in our framework is the asynchronous window. Figure 5.2 shows a conceptual design of bandwidth partitioning. The top-level reservation is the asynchronous window, which we divide into two partitions, i.e., for two applications. The reservations at this level represent the root servers for given hierarchies in our model (i.e., partition 1.1 and 1.2 in Figure 5.2). Then, for an application with multiple components that need mutual isolation, we can partition a reservation further. However, need for partitioning in more than three layers is seldom found in practice.

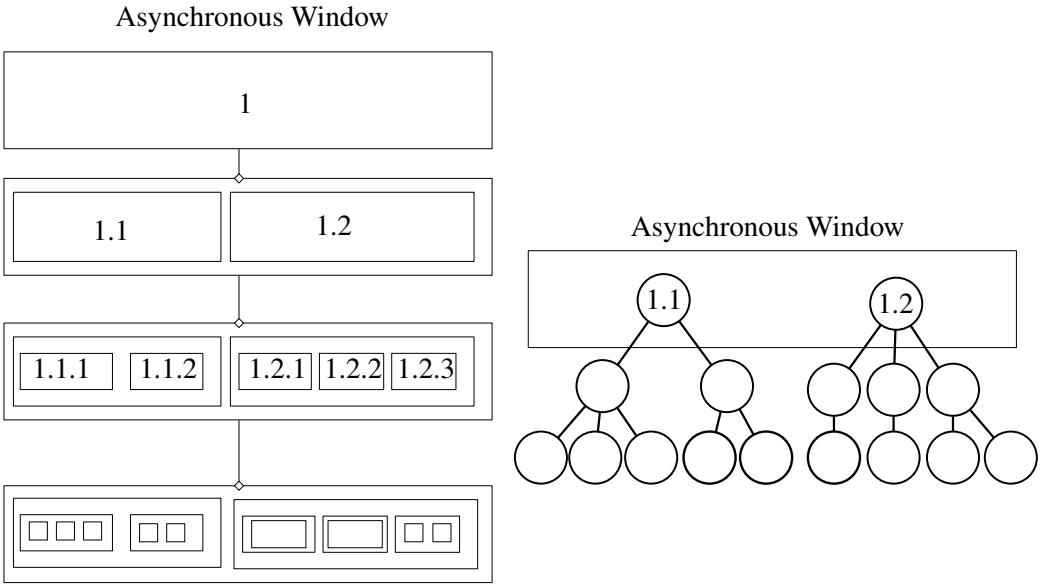


Figure 5.2: Partitioning the available bandwidth at different levels to provide mutual isolation across multiple applications in the system

We assume that the system consists of a set of n nodes connected to one switch. The switch has p ports, and the ports are full duplex. Each node is connected to only one port, and the input to the switch that receives streams from the node is called *uplink*, and the output from the switch to the node is called *downlink*.
The management of the servers' capacity is performed according to the polling server technique, as explained in Section 3.1.1, i.e., the server capacity is replenished periodically every predefined period, and when a server is scheduled, and the required load is less than its capacity, the unused capacity is discarded [26]. We also assume that the schedulers in all servers use the Rate Monotonic (RM) scheduling policy [103].

The server hierarchy that manages all streams originating from one source node and going to the same destination node is referred to as the *Independent Server Hierarchy* (ISH). Figure 5.3 shows a simple example of an HSS where one ISH manages all streams that will be directed to the Node 2 from Node 1, i.e., if Node 1 sends streams to m different nodes, then the master node should prepare m ISHs to manage the communication originated from this node. Note that, the maximum number of destination nodes that a node can send messages to is limited to $p-1$ (i.e., $1 \leq m < n \leq p$). In Figure 5.3, D_i is the destination for streams managed by an ISH and refers to the *downlink* in the Node i, whereas U_i is the source node producing the streams and refers to the *uplink* in the Node i.

In the HSS architecture, the interface associated with each server abstracts the resource requirements of its children. Consequently, we can represent the resource requirement of each ISH by the period and budget of its root server. Scheduling of the ISHs is thus easier since we do not need to consider the details of each ISH, but only its interface parameters.

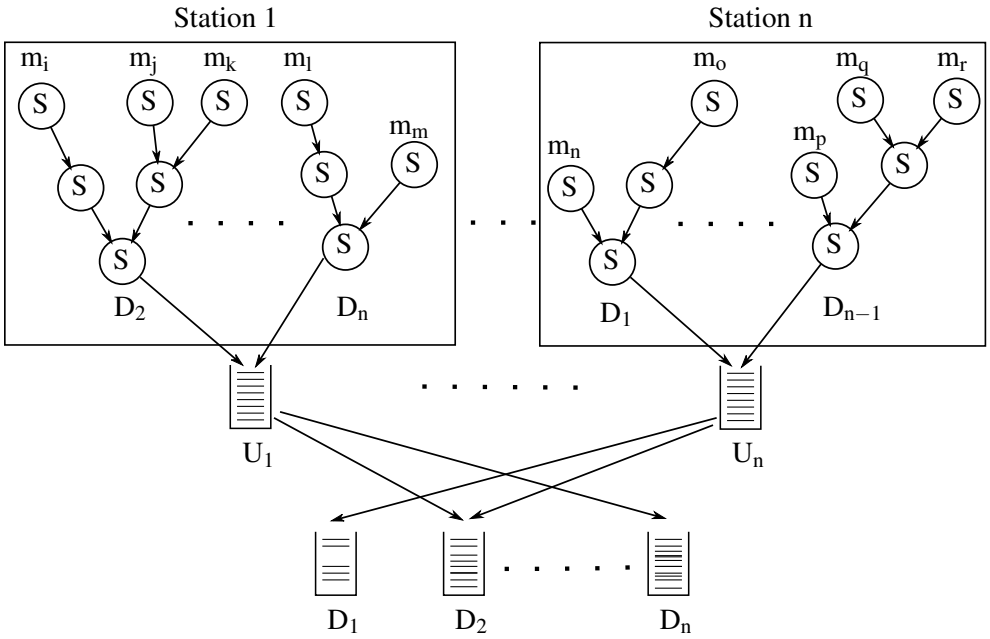


Figure 5.3: Hierarchical Server Based Scheduling (HSS) architecture

5.1.2 Servers and streams model

In our work, we use the HSS architecture to manage the asynchronous streams only. Asynchronous message streams (*AS*) are modeled using the sporadic real-time model in (5.1), where C_x is the message transmission time of a stream AS_x instance, $Tmit_x$ represents the respective minimum interarrival time and D_x the deadline. It is assumed that a message stream instance may generate several packets, which have a size comprised between $Mmin_x$ and $Mmax_x$. P_x identifies the parent server, i.e, the server to which the stream is connected to and RT_x is its computed response time.

$$AS_x = (C_x, Tmit_x, Mmax_x, Mmin_x, P_x, RT_x, D_x) \tag{5.1}$$

A server Srv_x is characterized in (5.2) by its capacity C_x, replenishment period T_x, deadline D_x equal to period, and a few data extracted from the set of children components, either servers or streams, namely the maximum and minimum packet transmission times ($Mmax_x$ and $Mmin_x$, respectively). Moreover, the server Srv_x is associated with a parent server P_x and a corresponding computed upper bound response time RT_x. Despite the similarity between the characterization of servers and streams, there is a fundamental difference since only streams imply actual transmission time that uses the capacity of the respective servers. Servers merely characterize a reservation of the network resource.

$$Srv_x = (C_x, T_x, Mmax_x, Mmin_x, P_x, RT_x, D_x) \tag{5.2}$$

In the remainder of this chapter we will refer to both streams and servers as components, in an integrated way.

5.1.2.1 An illustrative example

We present an example to highlight the difference between flat and hierarchical reservations. Imagine that we build soft real-time communication services for a set of applications using FTT-SE. Thus, the asynchronous window is the available periodic resource to each application in our system. Consider two applications $\mathbb{A}$ and $\mathbb{B}$ where $\mathbb{A}$ comprises message set $\{m_{35}, m_{36}\}$, and $\mathbb{B}$ comprises message set $\{m_{37}, m_{38}, m_{39}, m_{40}, m_{41}\}$. Assuming that, enough resource is available to meet the periodic demands set forth by each application, we study in which ways the hierarchical reservations improve on the flat reservations. Now, imagine that these two applications or the seven messages are being scheduled through reservation-based scheduling, following the setting depicted in Figure 5.4 (left) for flat reservations or Figure 5.4 (right) for hierarchical reservations. Regarding temporal isolation, respective server interfaces must guarantee isolation across different messages irrespective of the way reservations are managed, i.e., flat or hierarchical.

We can see the results of message schedule with flat reservations in Figure 5.5 (a), whereas the result of message scheduling with hierarchical reservations is shown in Figure 5.5 (b). In these figures, the vertical axis shows the EC in which the message is scheduled for transmission. The label on the box is the number of packets scheduled in that EC.

Flat reservations inherit the parameters (C, T) of the respective messages. Schedule building is confined only by the window size, and thus when further packets cannot fit inside the transmission window, scheduling is postponed to the following EC. Nevertheless, each reservation has

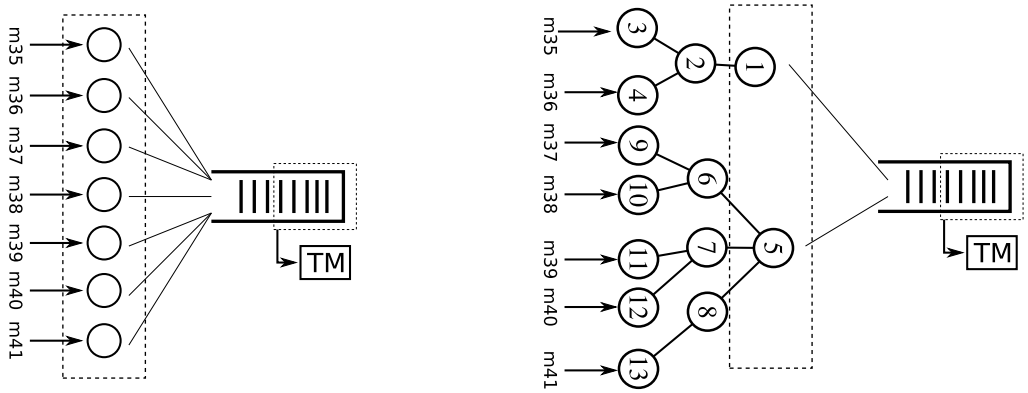


Figure 5.4: Flat and hierarchical reservations

the capacity to schedule a complete instance of the respective message. For hierarchical reservation, firstly, root servers are prioritzed according to their period, and then, at each internal level, servers are selected based on priority and remaining capacity until the leaf server when messages are scheduled. Hierarchical reservations, are confined, primarily, by the tight capacity of the respective reservations.

In the flat model, applications receive a faster response time. As noted, flat servers provide more bandwidth in each scheduling instance when compared to the root servers in hierarchies. These root servers further divide the bandwidth among their child nodes, and consequently, the available bandwidth at the leaf servers (where streams are connected) is much smaller. However, the message set is the same in both cases. For flat reservations, the respective server has enough capacity to schedule an instance of the message stream. And, thus, all the streams are scheduled in the priority defined by the RM policy without being held on account of insufficient server capacity. On the other hand, interface parameters for hierarchical reservations are tight that guarantee schedulability closer to the deadlines. The messages comprise multi-fragments, and a message completes transmission when its last fragment is scheduled. A message gets transmission opportunity only in certain ECs which depends on server availability. In particular, this example shows the time until one instance of each message released at the beginning of the system execution (started in EC 0) completes its transmission. In this example, we show the timeline until $t = 55$ which indicates the EC of the latest transmission i.e., that of m_{39}. However, during this interval, some messages could be released and complete their transmissions multiple times e.g., m_{37}, m_{41}. The purpose of this example is, however, to highlight some difference between the two reservation scheduling models. The hierarchical reservations lend a better control to the designer, smooth out the traffic, and thus might provide a better transmission opportunity for other traffic. For instance,

– if we are interested in prioritising the applications that are composed of several messages, this can easily be realised by setting the desired priority at the root server of the hierarchy.

– in a similar way, by creating branches, messages that belong to different sub-parts of the application are isolated.

In general, a hierarchical reservation may multiplex several messages for accessing the resource. Flat reservations, on the other hand, represent a single priority-based queue where each message has to compete against all the rest for access to the resource.

5.1.3 Scheduling model and execution

In this section, we explain the scheduling model and how it is executed by the master node to schedule all asynchronous traffic at each EC while at the same time respecting the underlying FTT-SE communication protocol. Figure 5.6 shows how the hierarchical scheduling framework is integrated with the FTT-SE master scheduler. The system consists of a database of ready aysnchronous streams AS_x as well as a repository of hierarchical servers. The FTT-SE master uses a global named queue ART_S_QUEUE to handle the ready asynchronous traffic and build the EC-schedules. Some of the streams are mapped in server hierarchies. Asynchronous messages that are not associated to a server hierarchy have, by FTT-SE construction, an associated flat server (Chapter 4). Flat servers compete with root servers of ISHs for access to the asynchronous window. In this case, FTT-SE uses a simple protection mechanism that enforces a sporadic arrival behavior [85] in respective streams. Concerning the messages mapped in server hierarchies, the scheduling is carried out in the following manner.

At every EC, the master node inserts all root servers of ISHs, that have non-zero remaining capacity, in a ready queue according to the Rate Monotonic scheduling policy. Then, it picks one server at a time starting from the head of the queue (highest priority) and then it schedules its internal children servers and messages based on the servers capacities and the size of the asynchronous window. This operation is repeated for all servers in the ready queue in decreasing priority order until all servers in the ready queue are covered or no more messages can fit in the EC. The scheduled messages are encoded in the TM for transmission (EC-schedule).

When a root server is selected to be scheduled, it picks a child server that has the highest priority and non-zero remaining capacity. The selected child server applies its scheduling algorithm to select the highest priority child server from its local servers and the selection process is repeated until a leaf server is reached where asynchronous streams are available. The maximum amount of transmission of the streams associated to the scheduled leaf server is limited by the minimum of remaining capacities of all the servers along the path from this leaf server to the root server and also by a fitness function. A specific fitness test is used to check how much transmission can be allowed such that in the worst-case scenario, the packets can fit in a given EC. The fitness function keeps track of the transmissions in each link and in each EC using bins with limited capacities that represent the asynchronous window, one for the uplink (link that connects the source node to the switch) and another for the downlink (link that connects the destination node to the switch), see Figure 5.3. Note that when the scheduler adds a message to a bin associated with a downlink in a switch, it takes into account the additional delay imposed by the switch on the message and also the maximum jitter that the interfering messages in the downlink can have. If the message under test fits in both bins totally or partially (the number of packets that can fit in the EC is less than the total number of packets) considering all the other messages that also fitted before in the respective

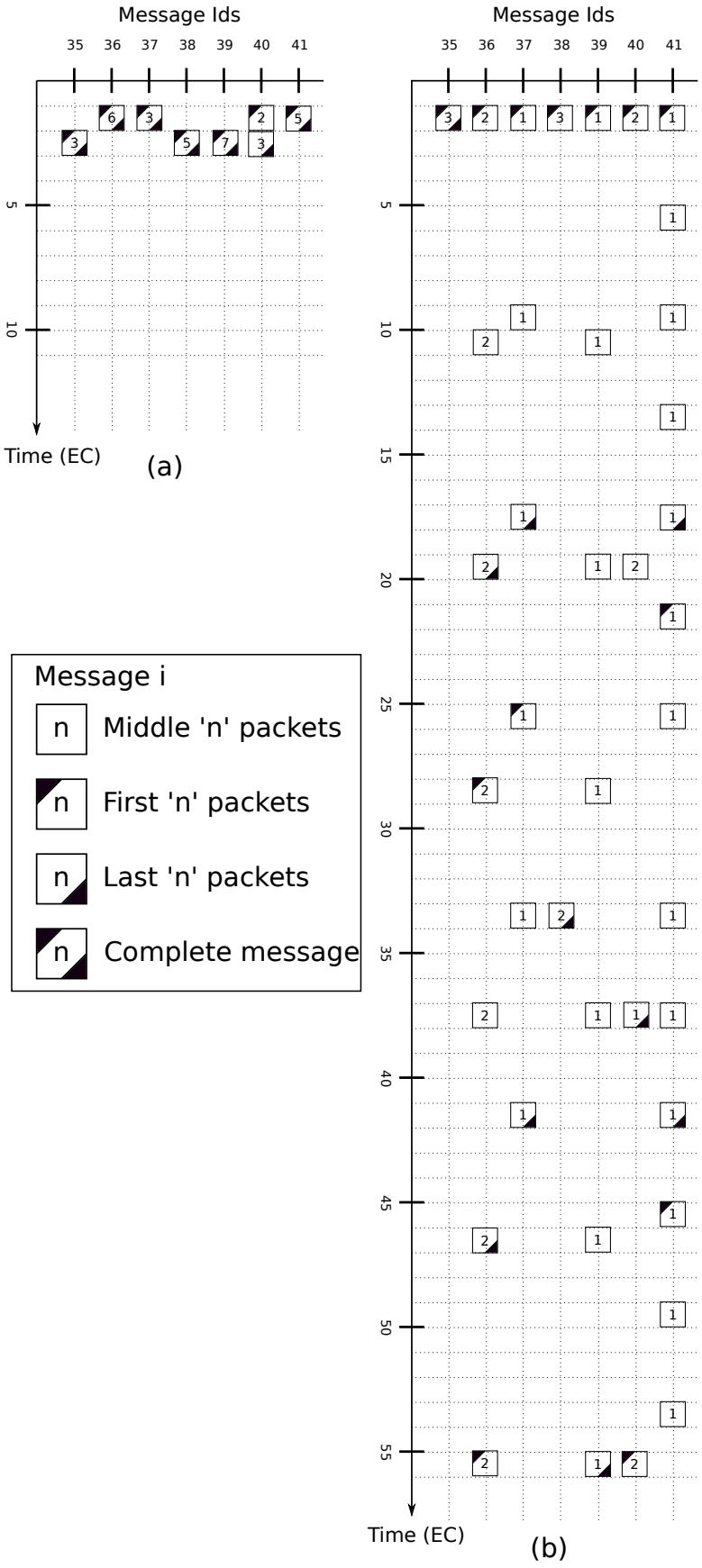


Figure 5.5: Message scheduling with flat reservations (a), and with hierarchical reservations (b)

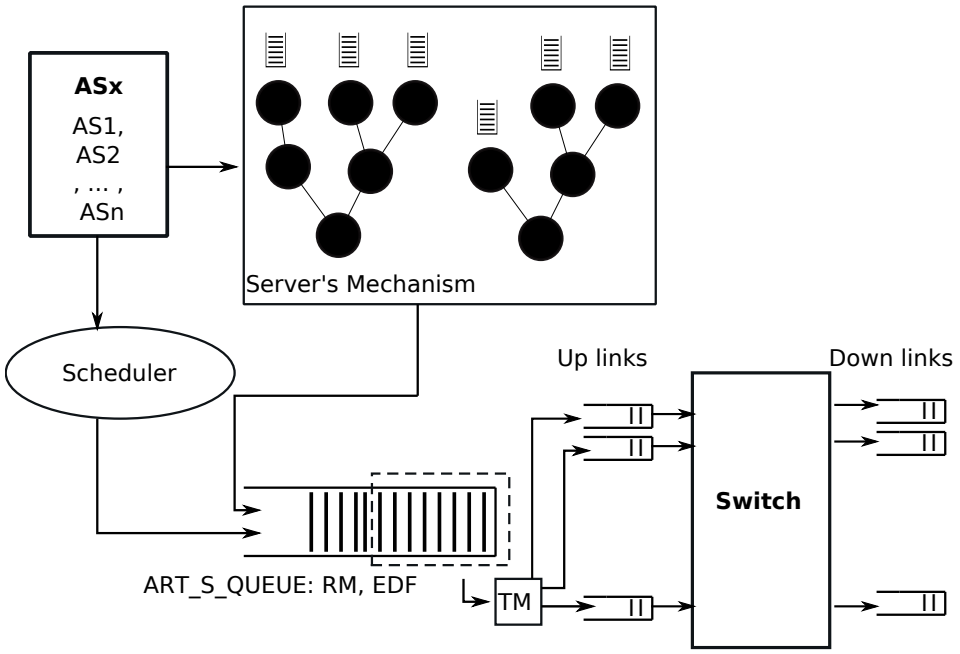


Figure 5.6: The scheduling model

bins, then that message is added totally or partially to the EC-schedule for the next EC to be later encoded in the TM.

The bandwidth consumed by the message (amount of scheduled transmission) is discounted from the capacities of all the servers on the path from the root server to the leaf server. If the root server has still some remaining capacity, it continues traversing the server tree to schedule other streams, until the capacity of the root server is exhausted or there is no more load to be scheduled. Since we are using the polling server, then any unused capacity will be discarded from the remaining capacity of the scheduled servers. This rule is applied for all servers in each ISH that are scheduled and have load (transmission request) from their children, less than the offered bandwidth (capacity) from the parents servers.

Note that assuming the number of nodes is n and each node may have up to $(n-1)$ ISHs then the upper bound on the total number of ISHs is thus $n \times (n-1)$ and in each ISH, the scheduler may visit all servers in the hierarchy. Let s_{max} denote the maximum number of servers in an ISH, then the scheduling algorithm may check $s_{max} \times n \times (n-1)$ servers at most in each EC.

5.2 Schedulability Analysis

In this section we present the schedulability analysis required by the admission controller that is used to verify whether change requests to the server hierarchy including adding, removing and changing the parameters of the components (servers and messages), are feasible, i.e., if all the components will meet their deadlines after applying the changes.

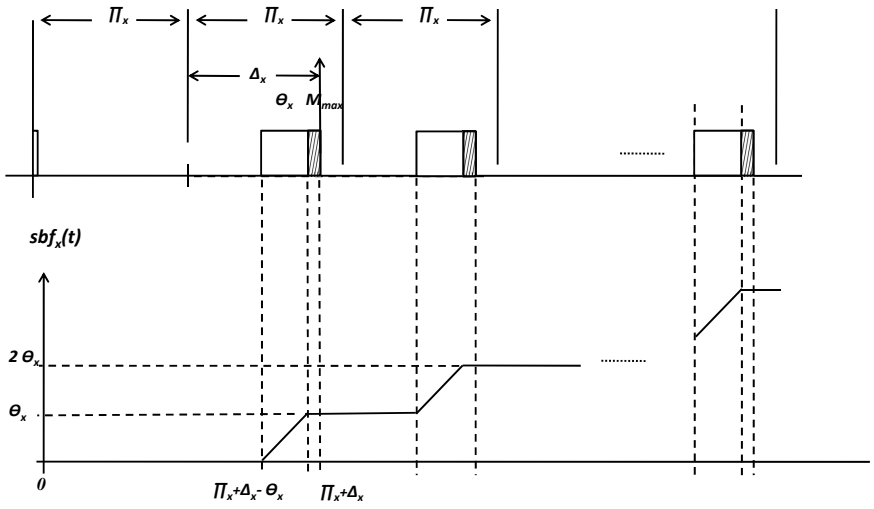


Figure 5.7: The supply bound function assuming the polling server.

We assume that the periods and budgets of servers are given (selected by the system designer) and it is required to verify that all messages finish their transmission before their deadlines and all servers can provide enough bandwidth to their respective children servers and/or frames. For this reason, we use the two phase approach presented in [129] to evaluate the schedulability algorithm. In the first phase, the values of M_{max} and M_{min} are evaluated (or updated after changes) for each server in the hierarchy starting from the leaf servers where they are directly connected to the streams, and propagating the maximum packet size and the minimum packet size to the parent servers and the parent of the parent server up to the root server. For each server, the values of M_{max} and M_{min} are equal to the maximum value of M_{max} and minimum value of M_{min} of all its children.

In the second phase, the response time of components (including the servers and streams) are computed starting from the root server of each ISH and then continue to its children then the children of the children down to the leaf servers and frames connected to them. The response time RT_x of a server/frame Γ_x is computed as follows;

$$RT_x = w_x + Mmin_x, \tag{5.3}$$
$$w_x = \text{ earliest } t > 0 : \mathbf{rbf}_x(t) = \mathbf{sbf}_{P_x}(t)$$

where $\mathbf{rbf}_x(t)$ is the request bound function of the server Γ_x that quantifies the maximum load

submitted up to instant t to the parent component P_x by the component itself together with the interference of higher priority components and it is computed using (5.4). And $\mathbf{sbf}_{P_x}(t)$ is the supply bound function associated to the parent component of Γ_x that computes the minimum bandwidth supply provided to its children at instant t which can be computed using (5.6). Evaluating the response time using (5.3) can be done using the algorithm presented in [129] with a bounded number of iterations.

$$\mathbf{rbf}_x(t) = IH_x(t) + C_x - Mmin_x \tag{5.4}$$

where $IH_x(t)$ is the interference from higher priority components and it is computed as follows;

$$IH_x(t) = \sum_{\Gamma_j \in hp(\Gamma_x)} \left\lceil \frac{t}{T_j} \right\rceil \times C_j \tag{5.5}$$

where $hp(\Gamma_x)$ is the set of components that have priorities higher than that of Γ_x and share the same parent server. However, this definition is valid only for all servers and messages except the root servers of ISHs. For the root server, evaluating Γ_x is different because at this level, the bandwidth is provided by the asynchronous scheduling window and the scheduling of messages of each ISH depends on the fitness function explained in the previous section. Note that the root servers abstract the bandwidth requirement of all associated messages and therefore each root server can be modeled as a message with the transmission time equal to the root server capacity and the period equal to the root server period. As a result, the schedulability analysis of FTT-SE based on utilization bound presented as in [86], that takes the fitness function into account, can be used to check the schedulability of the root servers. However, this analysis does not provide the response time of the root servers that will be used in the analysis of their children servers, as will be shown later in this section. Nevertheless, we will explain how to define $hp(\Gamma_x)$ such that it takes the fitness function into account when evaluating the response time of root servers. Let us define HPU_{Γ_x} as a set of root servers that share the same source node as the root server Γ_x and have priority higher than Γ_x. Also let us define HPD_{Γ_x} as a set of root servers that share the same destination node as the root server Γ_x and have priority higher than Γ_x. Note that the fitness function considers the interference from higher priority messages that share the same source or destination nodes with the message under consideration and also the jitter that the interfering messages may have at the downlink. Then the set $hp(\Gamma_x)$ can be redefined for server roots as $\{\Gamma_j\} | \Gamma_j \in HPD_{\Gamma_x} \vee \Gamma_j \in HPU_{\Gamma_x} \vee \Gamma_j \in HPU_{\Gamma_{HPD_{\Gamma_x}}}$, i.e., it includes all root servers that have priority higher than that of Γ_x and share the same uplink HPU_{Γ_x} or the same downlink HPD_{Γ_x} or servers that may add a jitter to the interfering root servers (HPD_{Γ_x}) that share the same downlink $HPU_{\Gamma_{HPD_{\Gamma_x}}}$.

The above mentioned problem was not presented in [129] since the source nodes in that work send their messages independently on the status of the downlinks that are connected to the destina-

tion nodes. Another difference between the analysis of FTT-SE and the analysis presented in [129] is that we do not need to include the blocking that can be caused by lower priority components in the computations of $\mathbf{rbf}_x(t)$. The reason is that the beginning of the transmissions from all nodes are synchronized with the reception of the TM sent from the master node at each EC and all traffic that was supposed to be sent during the previous EC should have been received before the beginning of the EC. So it is not possible to have a case where a higher priority message being ready to be submitted just after a lower priority message has started its transmission.

Another important issue that should be considered in the analysis is the switch delay that the messages suffer from when they are forwarded to their destination. In this work, we assume that the transmission time of the messages include this delay.

To evaluate the $sbf_x(t)$, the explicit deadline periodic (EDP) resource model [142] is used which is characterized by $\Omega = (\Pi, \Theta, \Delta)$, where Θ is the units of the resource provided within Δ time units (deadline) and with period Π of repetition. This way, mapping to our framework, a server is defined as $\Gamma_x = (\Pi_x, \Theta_x, \Delta_x) = (T_x, C_x - Mmax_x, RT_{P_x})$. Note that the deadline is given by RT_{P_x}, i.e., the worst-case response time of the parent component. Also, according to FTT-SE, the server capacity is strictly enforced and thus overruns cannot occur. Consequently, idle time may appear at the end of each server instance whenever the capacity available is not enough to transmit the next packet. The maximum inserted idle-time that a server component Γ_x can suffer is upper bounded by the maximum packet transmission time managed by this server. The impact of the inserted idle-time is accounted by deducing *Mmax* from the supply function, i.e., $\Theta_x = C_x - Mmax_x$. The component that provide resources to the root servers is the asynchronous window which can be modeled as $\Gamma_0 = (\Pi_0, \Theta_0, \Delta_0) = (EC, LW - Mmax_0, LW)$. For the root servers, *LW* which is the length of asynchronous window, is provided every EC and since the provision time of *LW* is constant every EC then the deadline will equal to the *LW*.

Now the supply bound function, assuming a polling server type, can be evaluated as follows;

$$sbf_x(t) = \begin{cases} b\Theta_x + \max\{0, t - a - b\Pi_x\}, & t \geq \Delta_x \\ 0, \quad otherwise \end{cases} \tag{5.6}$$

where

$$a = (\Pi_x + \Delta_x - \Theta_x), \quad b = \left\lfloor \frac{(t - (\Delta_x))}{\Pi_x} \right\rfloor \tag{5.7}$$

Note that, a in the previous equation (5.7) represents the maximum time that a server may not get any resources from its parent server. Using a polling server, the worst case scenario can happen when a server is scheduled at the beginning of its period and the server does not have any ready transmission request so the budget will be discarded, and a transmission request arrives just after that, and the server is scheduled as late as possible in the consecutive period (see Figure 5.7). Note

that for root servers of all ISH, this scenario will never happen because the transmission request is queued from the previous EC and it is ready at the beginning of each EC, therefore we can remove Π_x from (5.7) that computes a. Finally, to compute the end-to-end delay of the messages, two extra ECs should be added to the response time of messages to consider the delay caused by the asynchronous traffic signaling mechanism.

5.3 Evaluation

In this section, we validate our proposed analysis with tests on a real prototype, and with a master scheduler simulator which uses the server-based scheduling algorithm presented in Section 5.1.3. Besides, we verify the property of temporal isolation among message streams that share the network.

5.3.1 Experimental setup

We consider a single switch, three slave nodes (stations) (A, C and D) and one master node. Figure 5.8 shows the experimental setup. Station A contains two applications, and each application sends asynchronous traffic to another station. Each application has sub-applications and sub-sub-applications, and they are managed in the master node using one ISH for each application as shown in Figure 5.9. Station C has two applications, one sends messages to Station A and the other sends to Station D while Station D has one application that sends messages to Station A (the details are shown in Figure 5.9, where "D_X" in the figure indicates which destination node the traffic of each application (ISH) will be forwarded to, i.e. X= A or C or D). The total number of messages that are communicated through the network is 12, and 22 servers are used to manage the transmission of these messages. We set the value of $EC = 10ms$, and the maximum packet transmission time $Mmax_x = 88\mu s$. The length of the asynchronous window is approximately $LW = 50\%$ of the EC. The details of the servers parameters and the messages parameters are shown in Table 5.1 and Table 5.2 respectively.

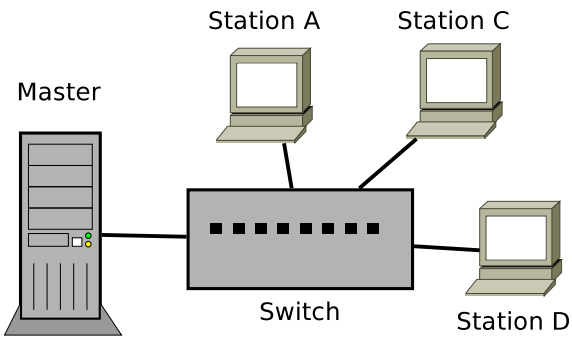


Figure 5.8: The experimental setup, with three slaves and the master node.

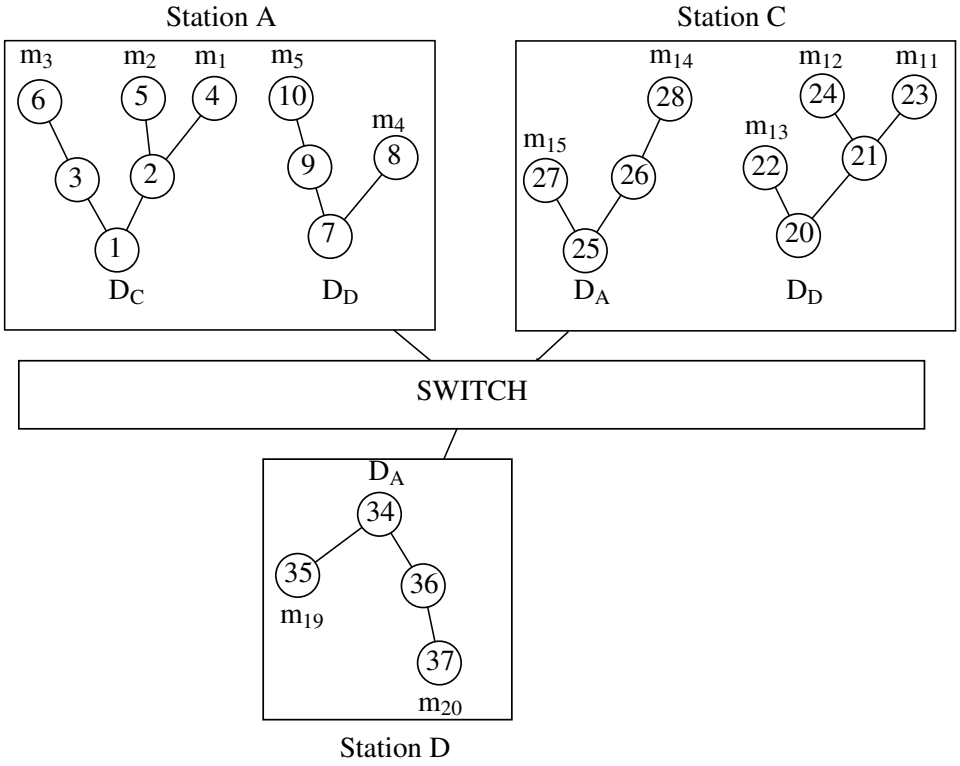


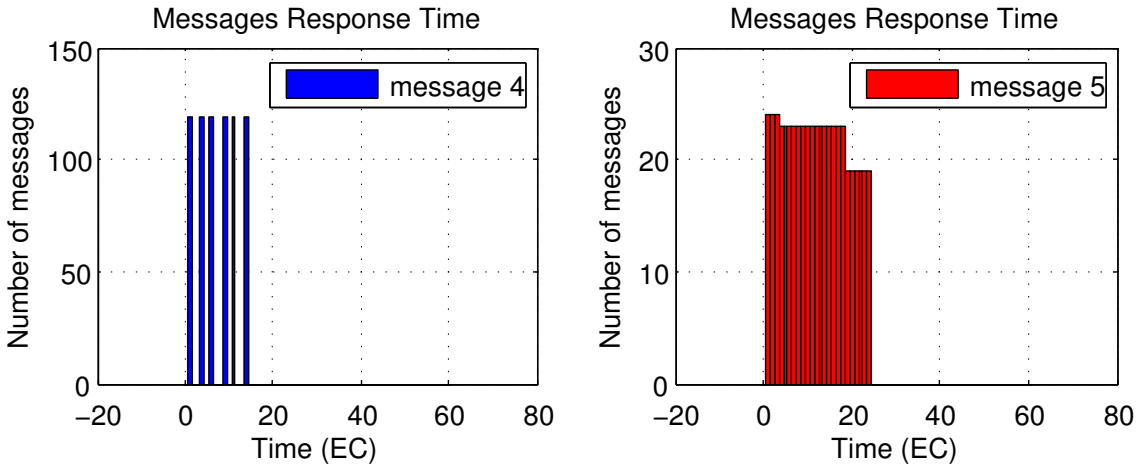
Figure 5.9: Independent Server Hierarchies (ISH) prepared at each source station

5.3.2 Analysis results vs. observation

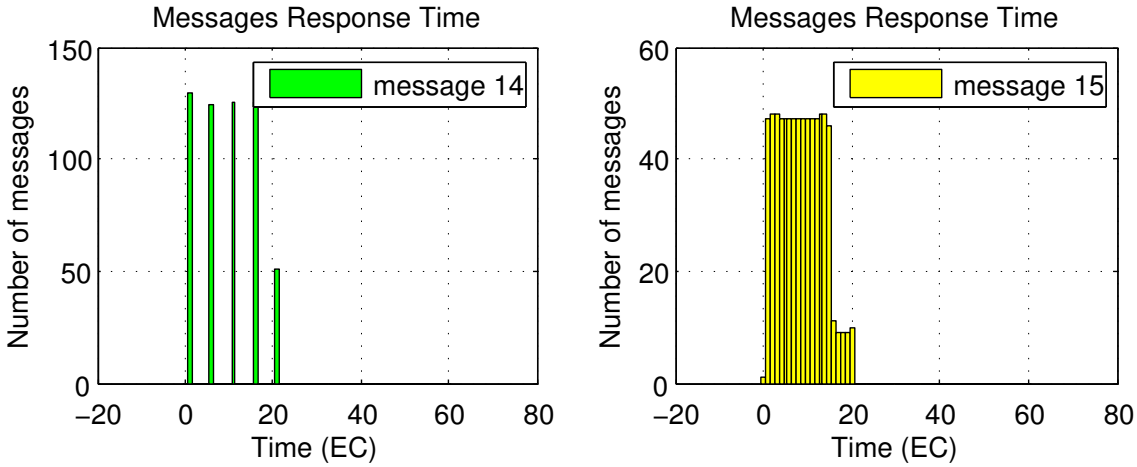
In this experiment, we compare the response times of messages that were measured from the implementation with the response times calculated using the analysis presented in Section 5.2. In the implementation, we activate all asynchronous messages periodically (with periods equal to their minimum inter-arrival times) to increase the load imposed on the network. Then we compute message response times, and the response time of each message is measured after receiving the signal from the slave station and right before dispatching the trigger message in the master node. Figure 5.10 shows the periodic arrival patterns and corresponding response times of certain selected messages. Table 5.3 shows the measured response time (RT measured) and the calculated response time (RT calculated). We can see that the estimated response times have higher values than the measured response times, which is expected. The reason for this difference is that measuring response times might not show the worst case scenario since it depends on the activation pattern of the messages and this periodic activation pattern does not generate the worst-case response time.

5.3.3 Checking temporal isolation

In this experiment, we show that temporal isolation is achieved i) among messages streams in the same ISH, i.e., belonging to the same slave station, and ii) among different network ISHs in the same source node and also iii) from different nodes. We also iv) show the effect of adding a new server in an ISH on the response time of messages in the same ISH.



(a) m_4 and m_5



(b) m_{14} and m_{15}

Figure 5.10: Message response times with periodic arrival patterns

Table 5.1: Server parameters for stations

Station	ISH	Id(x)	$C_x(\mu s)$	T_x (EC)
Station A	U_A-D_C	1	1998	4
		2	1470	8
		3	440	10
		4	440	16
		5	778	17
		6	352	20
	U_A-D_D	7	1682	6
		8	528	15
		9	902	12
		10	714	23
Station C	U_C-D_D	20	2024	4
		21	1936	8
		22	880	16
		23	792	16
		24	704	18
	U_C-D_A	25	1410	5
		26	880	10
		27	704	16
		28	704	22
Station D	U_D-D_A	34	1848	6
		35	880	12
		36	704	13
		37	704	24

Table 5.2: Specification of message parameters

ISH	Id(i)	C_i (μs)	$Tmit_i$ (EC)
U_A-D_C	m_1	528	50
	m_2	440	40
	m_3	176	45
U_A-D_D	m_4	352	35
	m_5	528	47
U_C-D_D	m_{11}	616	40
	m_{12}	440	37
	m_{13}	440	35
U_C-D_A	m_{14}	440	45
	m_{15}	528	33
U_D-D_A	m_{19}	528	28
	m_{20}	440	50

Table 5.3: Messages measured and calculated response times.

Id(i)	RT measured (EC)	RT calculated (EC)
m_1	32	45
m_2	14	30
m_3	17	35
m_4	15	23
m_5	25	43
m_{11}	10	29
m_{12}	21	31
m_{13}	17	25
m_{14}	22	38
m_{15}	20	27
m_{19}	18	20
m_{20}	24	43

Considering case i) above, we study the temporal behaviour of an ISH in station A containing messages m_4 and m_5 for station D. First, we consider periodic activation of m_4 and m_5 and see the response times. Figure 5.10 shows the results, typical uniform distribution of response times with polling servers.

Let us now select to burst m_5, which is transmitted in the same ISH, by sending transmission request every EC. We have measured the response time of m_4, and it shows the same behaviour as the first case when all messages are activated according to their specifications (see Figure 5.11). In the experiment, we also note that the response times of all other messages in the network are not affected.

Now we select messages that belong to the ISH that share the same source node. In this case, we have messages m_1, m_2 and m_3 that send requests to transmissions every EC. The results that we measured show that the response time of m_4 does not change.

In the third step, we burst the activations of m_{11}, m_{12} and m_{13} which share the same destination as m_4 and again the worst case response time measured does not change.

Finally, we test the ability of the protocol to adapt after a change request. In this case, we assume that the system requests to add a new server to be connected to server 10 in the station A. The server parameters are $T_x = 25EC$, $C_x = 178\mu$s and this server will manage the communication of a message with the following parameters $D_x = Tmit_x = 50EC$, $C_x = 88\mu$s. The admission controller will first check if this server can be added without violating the time requirements of all other messages. This is done by applying the analysis presented in this chapter and since the priority of the new server will be less than the priority of server 10 which share the same parent, it is only required to check the schedulability of the new server by obtaining its response time and then check the schedulability of its message which is passed in both cases. Then the server is added to the network, and we measure the response times of all messages and compare them with the results of Experiment 1, which shows the same response times. This experiment shows that

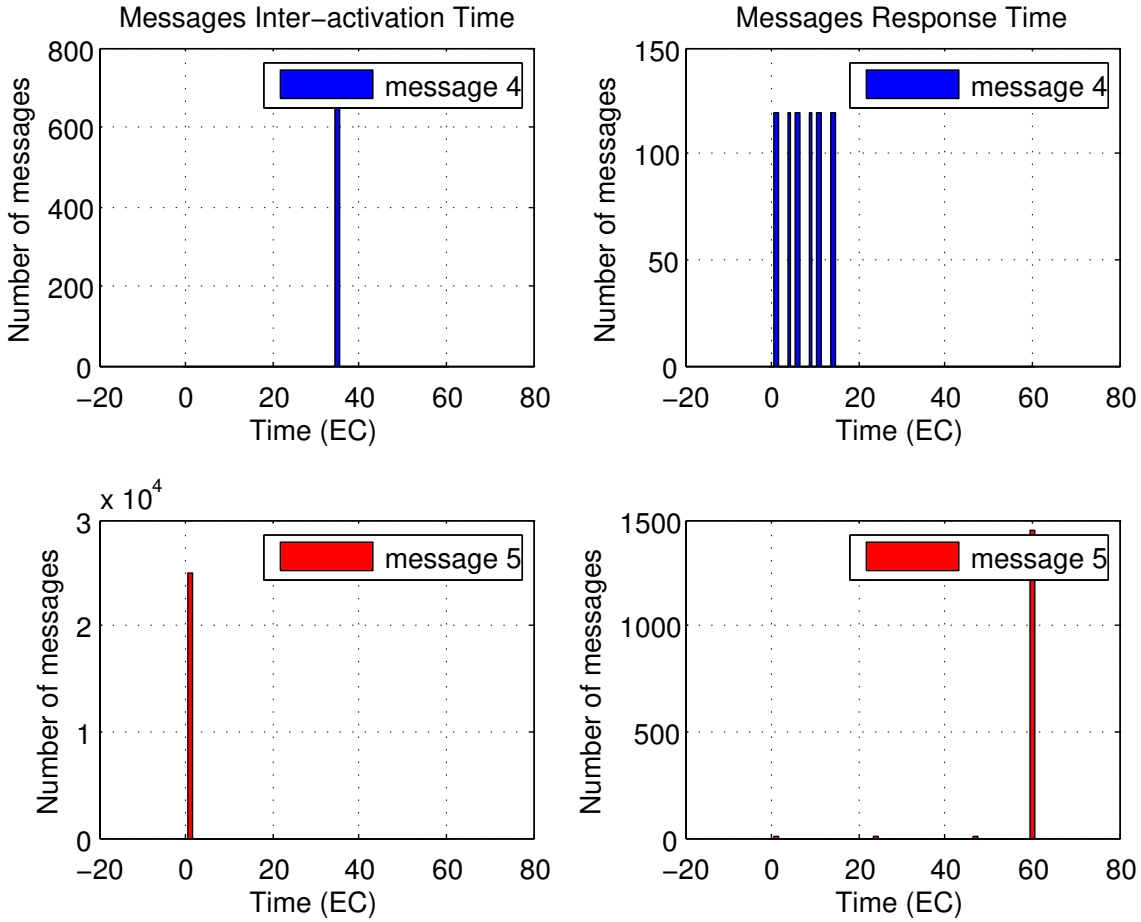


Figure 5.11: Arrival pattern (left) and response times (right) of messages m_4 and m_5 when message m_5 has bursty activations.

the temporal isolation is kept for all messages even if other messages are misbehaving and also in case of adding new messages. In addition, it shows that when we add a new server, we do not need to re-calculate the response times of all servers and messages in the network, it is enough to check the affected parts in the ISH that a server will be added.

5.3.4 Verifying temporal isolation with random simulations

Using the simulator reported in Chapter 8, we generate several system configurations at random and verify the property of temporal isolation among message streams. For these experiments, we consider 3 slave stations, and a maximum of 2 ISHs per slave station. The duration of the EC is 3 ms. The duration of the asynchronous window is 1377 μs and maximum sized packet transmission time (MTU) is 88 μs. Message size is chosen randomly between 1 and 7 MTU packets and its minimum inter-transmission time is chosen randomly between 10 and 70 ECs. Across different simulation runs, the number of ISHs and hence the total number of servers and message streams are different. In a simulation trace consisting of 5000 cycles, we observe the message response times when messages are scheduled in an HSF under a *regular* mode and an *induced congestion*

mode. In the *regular* mode, the activation pattern for each message in the set is periodic whereas in the *induced congestion* mode, we choose at random 5 messages in the set such that these messages deviate from their periodic activation pattern and generate 3 times higher load instead than with the *regular* mode.
In the following, we report an example system configuration generated by the simulator (Figure 5.12). We have 3 stations connected to the switch. Station 1 has two applications, whereas stations 2 and 3 each have one application. The applications in station 1 send data to the other two stations, whereas stations 2 and 3 send data to each other. There are 19 total messages and 32 servers. The parameters of the message set are given in Table 5.4 and servers parameters are given in Table 5.5.
Firstly, we schedule the given message set under *regular* mode. Figure 5.13 shows the message response times. The blue lines extend from message minimum response time until its maximum observed response time. The red dots indicate the message periods (equal to message deadline). We can see that, with a periodic activation pattern, the message set is schedulable. Next, we induce congestion in the network; we randomly pick 5 messages in the set and increase their offered load. For the same message set, we ran the simulation 6 times, each time causing burst within different messages. This way, we can reflect increased load in different parts of the hierarchy. Results are shown in Figure 5.14. We can verify that property of temporal isolation is observed among different streams. In particular, the bursty messages are indicated by dashed vertical lines that extend beyond their deadlines (deadline == period and maximum message period in the simulations is 70). Such messages achieve much longer response times during the observed simulation trace. However, for better readability, we show results up to 100 cycles on the vertical axis. By comparing each of these graphs to the one generated with *regular* mode, we can see that response times of all the messages which respect their periodic activation pattern remain unaffected by the burst in other messages.

5.4 Summary

In this chapter, we showed how multi-level hierarchical scheduling could be efficiently deployed using FTT-SE. We presented a response-time based schedulability analysis that takes into consideration the FTT-SE protocol together with the multi-level hierarchical server-based scheduling. In addition, we presented a prototype implementation that validates the analysis of the architecture proposed in this chapter. Furthermore, the results obtained from the prototype implementation verify the property of temporal isolation among message streams that share the network bandwidth in case of overload and changes in the hierarchical architecture. In particular, the results highlight the strong partitioning capabilities of our approach, with full temporal isolation across partitions in different branches of the hierarchy.

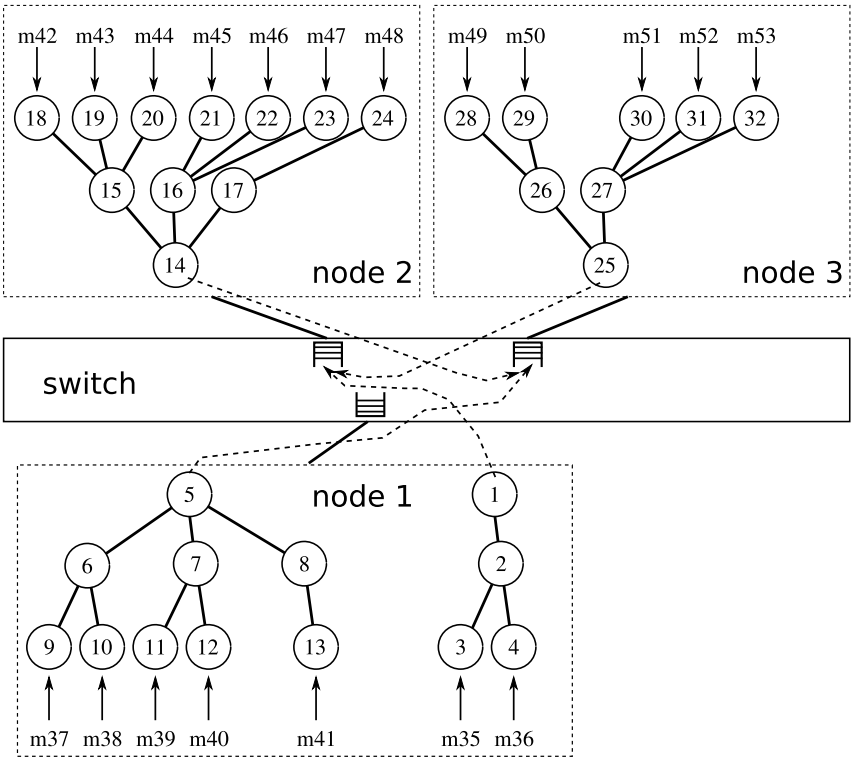


Figure 5.12: The experiment setup for random simulations

Table 5.4: Messages set parameters

Id(i)	Tmit (EC)	Size (μs)	Server
m_{35}	69	197	3
m_{36}	27	519	4
m_{37}	25	241	9
m_{38}	65	371	10
m_{39}	64	588	11
m_{40}	55	375	12
m_{41}	20	411	13
m_{42}	10	223	18
m_{43}	49	508	19
m_{44}	49	159	20
m_{45}	51	569	21
m_{46}	25	158	22
m_{47}	51	384	23
m_{48}	34	524	24
m_{49}	60	291	28
m_{50}	54	412	29
m_{51}	65	395	30
m_{52}	53	399	31
m_{53}	44	322	32

Table 5.5: Servers' parameters

Id(i)	T_x (EC)	C_x (μs)
3	69	264
4	9	176
2	3	440
1	3	440
9	8	88
10	32	264
6	8	352
11	9	88
12	18	176
7	9	264
13	4	88
8	4	88
5	1	704
18	3	88
19	24	264
20	24	88
15	3	440
21	25	352
22	25	176
23	25	264
16	25	792
24	5	88
17	5	88
14	1	1320
28	30	176
29	10	88
26	10	264
30	32	264
31	10	88
32	22	176
27	2	528
25	2	792

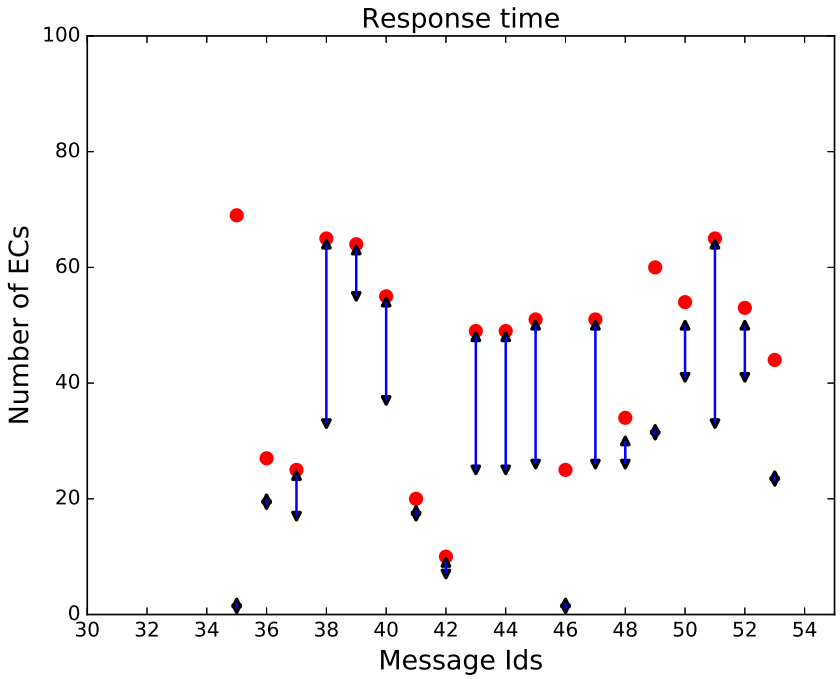


Figure 5.13: Message response time with *regular* mode. Red dots indicate the message period. Blue lines extend from message minimum observed response time (RT_{o-min}) to its maximum observed response time (RT_{o-max}) achieved in a simulation trace of 5000 ECs.

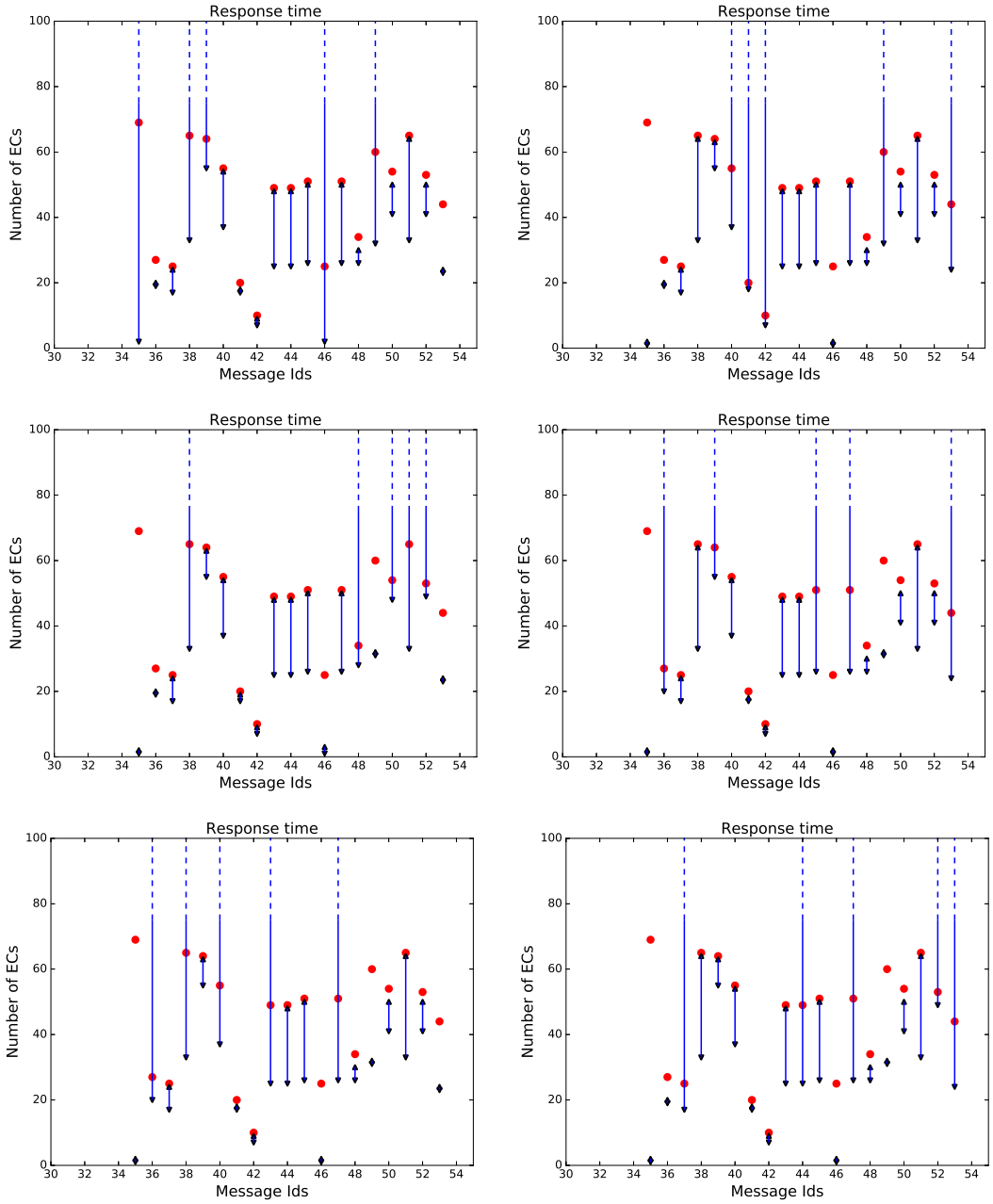


Figure 5.14: Message response times with *induced congestion* mode

Chapter 6

Supporting Hierarchical Reservations within FTT-SE using Sporadic Servers

Composability is an important property to build complex applications. In Chapter 5, we studied an important technique to achieve composability, particularly in the time domain, i.e., multi-level hierarchical server-based design. We used polling servers for reservation scheduling and replenishment. This chapter presents our work when we manage the hierarchical reservations using sporadic servers, instead, again implemented on Ethernet using FTT-SE. With the sporadic server-based policy, we can observe an efficient network bandwidth utilisation with shorter application response times.
In this chapter, we report the successful implementation of hierarchical server-based traffic scheduling in FTT-SE using sporadic servers, which decouple response times from the allocated bandwidth. Our specific contributions are:

- adaptation of the hierarchical server-based architecture to use the sporadic server model
- reference implementation of sporadic servers in FTT-SE
- experimental verification of low response time and temporal isolation between contending applications

The chapter is organised as follows: Section 6.1 describes the implementation aspects of the hierarchical server-based scheduling in FTT-SE using sporadic servers. Section 6.2 presents the experimental evaluation from our implementation showing its practical feasibility. Finally, in Section 6.3 we provide a summary of the chapter.

6.1 Implementing Hierarchical Sporadic Servers

We implement hierarchical sporadic servers as part of the FTT-SE scheduling algorithm. For implementation purposes, we need a mechanism that can manage the activations and capacity of servers following the particular server policy. For example, a polling server [26] must be activated

and its capacity replenished periodically whereas a sporadic server follows a sporadic consumption/replenishment model; it holds its capacity until it is requested, it can provide service immediately upon request as long as it has sufficient capacity available and it replenishes any amount of consumed capacity one period after it was requested (Section 3.1.3). We use the same HSF integration approach and scheduling model described in Section 5.1.3, but with a different scheduling algorithm due to using a different server policy.

6.1.1 Scheduling algorithm

In the case of polling servers, hierarchical scheduling is managed by having a queue of ready root servers that are triggered periodically, following which scheduling internal to the ISH takes place. With sporadic servers, two events can invoke the scheduling process:

(i) a message activation at the leaf server

(ii) replenishment for any server in an ISH

In both cases, the scheduling process once triggered begins at the root of the respective hierarchy. The server-based scheduling is outlined in Algorithm 2. The main scheduling function serverFunction taskes as input the root server node of the hierarchy and count of the current EC. The main tasks of this function include selecting an eligible leaf server node and then scheduling packets inside server queue, achieved respectively by the sub-routines findLeafServer (Algorithm 3) and schServerPkts (Algorithm 4).

Given the root server node in the hierarchy, Algorithm 3 will return the next available leaf server node in that ISH, and if the search does not return a leaf node, the function returns with the root node of the hierarchy indicating that ISH cannot schedule packets. In particular, the root server selects one of its ready children servers. A parent server checks two conditions to select one of its child servers. For the first condition, it checks the remaining capacity and state of the server. A server that is in ACTIVE state and has non-zero remaining capacity is eligible (see section 6.1.2 and 6.1.3). Following this, the child server with the minimum period is chosen, i.e., we use RM scheduling policy. Other algorithms can be used such as such as Earliest Deadline First (EDF) or Fixed Priority Scheduling (FPS) [112]. The selection functionality is encapsulated within routine pickNextBranch (refer Algorithm 3, line 7). Then the scheduled child server will select another child server and the same procedure will be repeated down the tree until we reach the leaf server which will finally schedule a message for transmission.

The routine responsible for scheduling packets (Algorithm 4) checks the capacity available to the server node (Algorithm 4, lines 3, 4). The amount of bandwidth given to the scheduled stream is the minimum among the remaining capacities of all parent servers along the path from leaf to the root. In particular, leaf server's own capacity is given in line 3 whereas minimum remaining capacity along server path is calculated using the routine getGuaranteedCapacity in line 4. Following this, the message from the queue of server node is retrieved (line 6), and the size of next packet is

computed (line 9). When a packet can fit within server's capacity, the consumed bandwidth by the stream is discounted from the remaining capacity of all the servers along this path (lines 15, 16). Also, the future replenishment instants are set for the corresponding servers (line 17). Finally, the server message queue is updated to reflect the state of served packets (line 18). If the server capacity is exhausted, the message is moved to the global ready queue ART_S_QUEUE (line 12) where the system level scheduler will schedule the transmission of packets that are already served. Then, the server becomes suspended until its capacity is replenished.

Taking into account that we only consider the aperiodic traffic which is scheduled through sporadic servers inside the asynchronous window of the elementary cycle, certain observations are in order. Firstly, this arrangement prevents high priority unbounded interference from periodic traffic. Secondly, sporadic servers are engaged in message scheduling under two events, upon message activation and server replenishment. Concerning message activation, in a given EC, several messages might be activated simultaneously. However, the mechanism deals with each message in turn; a given message when activated, instantly triggers the corresponding sporadic server. The server schedules messages within the limit of its capacity using an FCFS policy, barring the unavailability of time within the asynchronous window. Upon replenishment, sporadic servers may schedule enqueued messages following further rules (Section 6.1.2). In this scenario, the mechanism will also perform all pending replenishments that are due in that EC. It is noteworthy that, for the top level, all the sporadic root servers shall be checked in each EC for capacity replenishments. Once a root server is chosen, higher priority internal server(s) will take precedence using the available resource.

6.1.1.1 Example

We explain the scheduling algorithm through an example as shown in Figure 6.1 that shows different phases of message scheduling in HSF. In particular Figure 6.1 (a) depicts an ISH in the initial state. No messages have been active; the tuple represents $(\mathsf{period}, \mathsf{capacity}, \mathsf{state})$ for each server in the ISH. In Figure 6.1 (b), message m_{21} request arrives at server Srv_4, and corresponding path in the ISH from Srv_4 up to the root Srv_1 is activated. This is depicted with a bold line. Notice the change in the state of servers along the bold path. In Figure 6.1 (c), the request has been serviced partially. After service, the tuples have been updated along the bold path. Since, Srv_4 is out of capacity, its state is set as DEPLETED. Figure 6.1 (d) shows that another request m_2 arrives at Srv_5; corresponding path is activated, depicted here in bold. The state of servers along the bold path is also updated. Following this, in Figure 6.1 (e), m_2 has been completely served. We can notice how child servers share the capacity of the parent servers.

6.1.2 Replenishment management

An essential component of the scheduling model is the management of servers replenishment that is done as follows.

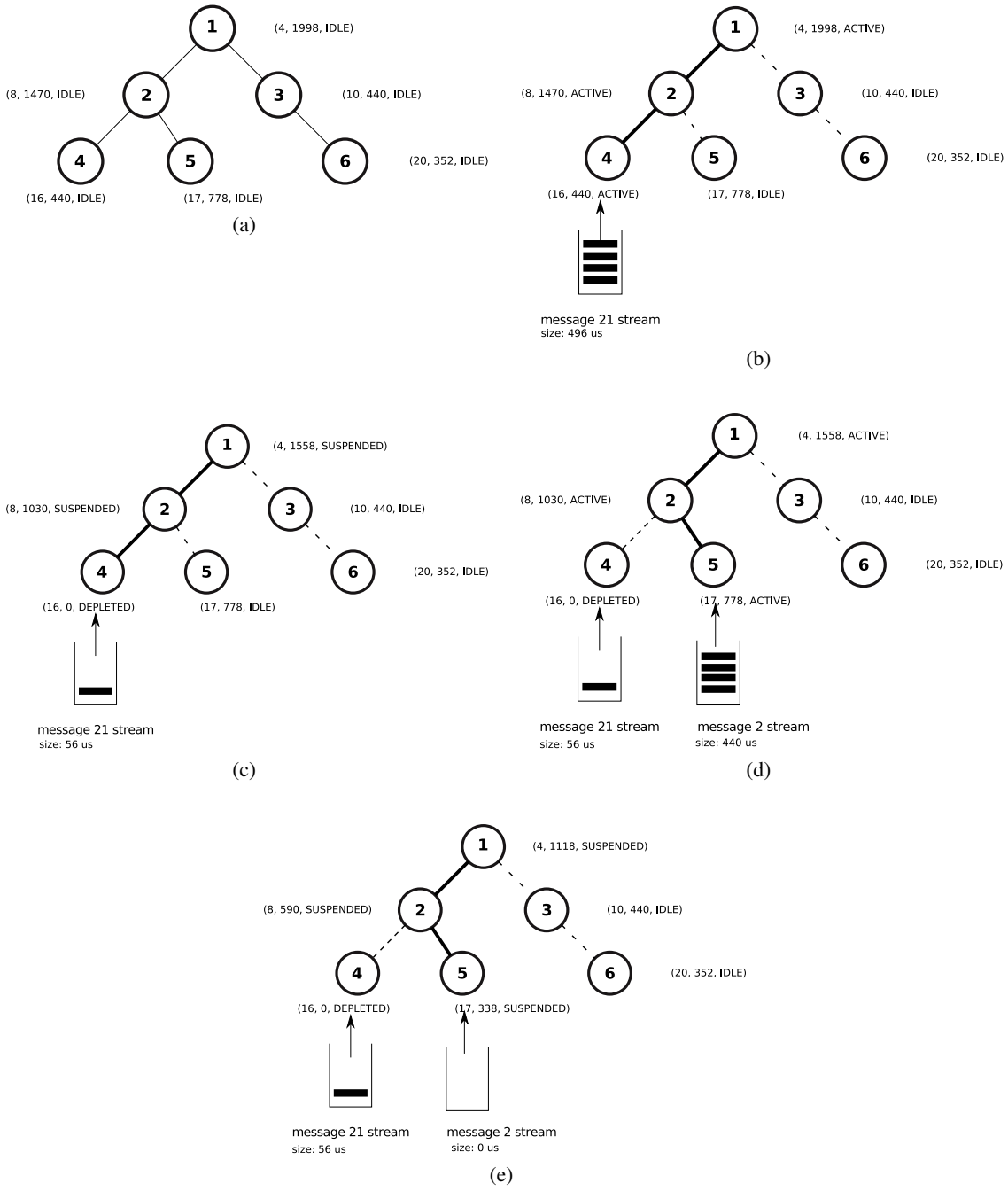


Figure 6.1: Example of scheduling messages with a sporadic HSF

Algorithm 2 Server based scheduling function

```
1:  function serverFunction(s_node,ec) ▷ server based packet scheduling in an ISH with server
    node s_node in EC ec.
2:      more_paths ← true
3:      while more_paths do
4:          s_node = findLeafServer(s_node)
5:          if s_node.s_level = L_ROOT then
6:              more_paths ← false
7:          else
8:              if s_node.s_level = L_LEAF then
9:                  id ← SV_Get_id(s_node)
10:                 period ← SV_Get_period(s_node)
11:                 msg_queue ← M_Queue_db_get_msg_q(id)
12:                 rep_queue ← M_Queue_db_get_rep_q(id)
13:                 schServerPkts(s_node,msg_queue,ec )
14:                 if s_node.capacity ≤ 0 then
15:                     s_node.f_status ← DONE
16:                 s_node.s_state ← SUSPENDED
17:             s_node ← s_node.parent
18:     return 0
```

attributes of the selected server node

Algorithm 3 Finding an eligible leaf server

```
1:  function findLeafServer(s_node)  ▷ Given the root server node s_node in an ISH, find an
    available leaf server node
2:      while true do
3:          if s_node.f_status = DONE then
4:              break
5:          if s_node.s_level = L_LEAF then
6:              break
7:          s_node ← pickNextBranch(s_node)
8:          if s_node.capacity ≤ 0 then
9:              s_node.f_status ← DONE
10:             s_node.s_state ← DEPLETED
11:             s_node ← s_node.parent
12:     return s_node
```

keep searching in all paths, return with a leaf node upon success or the root node on failure

Algorithm 4 Scheduling packets from the queue of selected leaf server s_node

```
1:  function schServerPkts(s_node,msg_queue, ec)
2:      pkts_served ← 0
3:      cap ← SV_Get_Capacity(s_node)
4:      cap ← getGuaranteedCapacity(s_node)
5:      while true do
6:          msg ← ready_queue_next_item(msg_queue)
7:          if msg = nil then
8:              break                                   ▷ Server queue is empty
9:          pkt ← get_next_fragment(msg)
10:         if cap < pkt.size then
11:             if pkts_served > 0 then
12:                 move_to_queue(ART_S_QUEUE, msg, pkts_served, pkts_sch)
13:             break                                   ▷ cannot serve more packets
14:         pkts_served ← pkts_served + 1
15:         update_capacity_on_path(s_node, pkt.size)
16:         cap ← cap − pkt.size
17:         set_rep_on_path(s_node, pkt.size, ec)
18:         ready_queue_decrease_item(msg_queue)
19:     return 0
```

update the state in ISH once a packet is served

We associate two queues with a server, a message queue and a replenishment queue. The message queue holds the application messages that must be handled by the server whereas the replenishment queue holds records for future replenishments of the server capacity. An entry in the replenishment queue is a pair of the type (C, t_{rep}). This entry is set when a request is served either partially or completely, thus, C is the capacity consumed and t_{rep} is the replenishment instant. t_{rep} is computed as: $t_{rep} = t + T_x$ where t is the EC in which C units of the request are scheduled for transmission and T_x is the server Srv_x replenishment period. When we schedule a replenishment at an instant t, we know the request size that can be served (partial or complete), i.e., which is within the limit of minimum available server capacity along the path. Thus, we replenish the server with only this amount exactly one period later. Hence, our implementation does not suffer from the premature replenishment effect. With this model in place, the scheduling algorithm for hierarchical sporadic servers works as follows.

At EC_i, we probe the replenishment queue for all servers. For each entry (C, t_{rep}) such that $t_{rep} == EC_i$, we replenish the server with C units. At this instant, we update the state as ACTIVE for every server in the respective ISH that has a non-zero capacity, and invoke the server-based scheduling serverFunction (Algorithm 2) from ISH root. This execution can schedule those messages that may have been left un-scheduled in the previous runs on account of insufficient capacities either at the leaf server or an intermediate/root server level. With the replenished capacity it may be possible to schedule such messages.

Listing 6.1: The structure representing a message item in the ready queue

```
typedef struct ready_queue_idx
{
        void *message_DB_pointer;
        void *rp;
        unsigned int packets_no;
        unsigned int to_be_served;
        struct ready_queue_idx *link;
}READY_QUEUE_IDX;
```

6.1.3 Processing message arrivals

The arrival of message stream activates the leaf server of the corresponding ISH as well as the servers along the path to the root. Processing an ISH involves verifying all servers in the path from that leaf (note that all servers receiving messages are leaves) up to the root. The scheduling is then invoked from the root server, calling serverFunction (Algorithm 2). So, the servers along the path become ready with new message arrivals. The capacity requested by a message, known when the message is scheduled for transmission totally or partially, is discounted from the remaining capacities of all the servers along the path up to the root. An entry is set in the replenishment queue of all the servers along the path indicating the future replenishment instant and the capacity to be added.

6.1.4 Handling message packets

An important part of the sporadic HSF implementation deals with the transfer of messages between server ready queues and ART_S_QUEUE global queue (Algorithm 4, line 12). Note that FTT-SE fragments large messages into small packets, which are individually scheduled and thus our servers also handle messages as bundles of packets that are then processed individually.

The following is the structure (Listing 6.1) that represents a message item in the ready queues where packets_no represents the total number of packets in the message.

We say a packet needs scheduling when it is already inside the global queue. We say a packet needs to be served when it is still in the server queue. The two fields namely packets_no, and to_be_served are used to denote the number of packets which are in one of the two states. Upon initialization, to_be_served (denoted pkts_served in Algorithm 4) is the same as packets_no. But, when a packet can fit in the server capacities, we only update to_be_served and not the packets_no. When no more packets fit the server capacities, those that did (were served) are copied to the global ART_S_QUEUE.

Hence, a message may be partially present in both the server queue and the ART_S_QUEUE at the same time. This happens when a message is partially served and is partially scheduled.

6.2 Evaluation

In this section, we perform an experimental validation of our implementation. Beyond showing the feasibility of the hierarchical sporadic servers framework, we also aim at demonstrating the superiority of these servers with respect to the polling servers used in [40] concerning their response times. Note that with sporadic servers messages can be immediately served whereas with a polling server a message has to wait until the next periodic replenishment to receive capacity. Moreover, we also aim at verifying the temporal isolation capabilities of our sporadic HSF. In our experiments, applications generate asynchronous traffic that is managed through hierarchical sporadic servers only.

6.2.1 Experimental setup

Our experimental setup consists of one switch, a master station and three slave stations A, C, and D. It is essentially the same setup used in Section 5.3 for the validation of the polling HSF. Thus, the results are comparable. Station A contains two applications. These applications have distinct components and data to send to the other two stations. The applications are managed in the master node through an ISH where one ISH represents one application. Station C has two applications; one sending traffic to station A and the other to station D. Station D has only one application that generates data for station A. This setup is shown in Figure 5.8. Regarding the ISHs that we prepare in the master station to manage the traffic generated by the applications, configuration of the ISH in station D is different than the one shown in Figure 5.9. Figure 6.2 shows this arrangement. D_X refers to the consumer of traffic generated by the respective applications; for example, in station A, D_B means that ISH will forward message $\{m_{21}, m_2, m_3\}$ to station B. The total number of messages in the system is 12 and 23 servers are used to manage these message transmissions. The parameters of servers and messages are given in Table 5.1 and Table 5.2 respectively.

We set the duration of $EC = 10ms$, and maximum packet transmission time is $Mmax_x = 88\mu s$. The length of the asynchronous window is approximate $LW = 50\%$ of the EC.

For faster response, we set server parameters empirically following simple rules, e.g., the capacity of the parent servers is more than the sum of their child servers. The capacity of each server is multiple of $88\mu s$ i.e., the packet transmission time. Normally, the capacity of each server is enough to schedule an instance of each of its children. This leads to short response times with periodic message activations. However, in the case of overload, the server capacities may not be enough for scheduling all jobs of a particular message stream. In this case, such messages remain in their queues until capacities become available. The server design problem for sporadic, deferrable or periodic servers, however, is orthogonal to the current work and it can be solved with techniques available in the literature [110]. However as noted later, we have done some work in this direction which we report in Chapter 7 for the design of polling servers, only, since they have specific synchronization requirements.

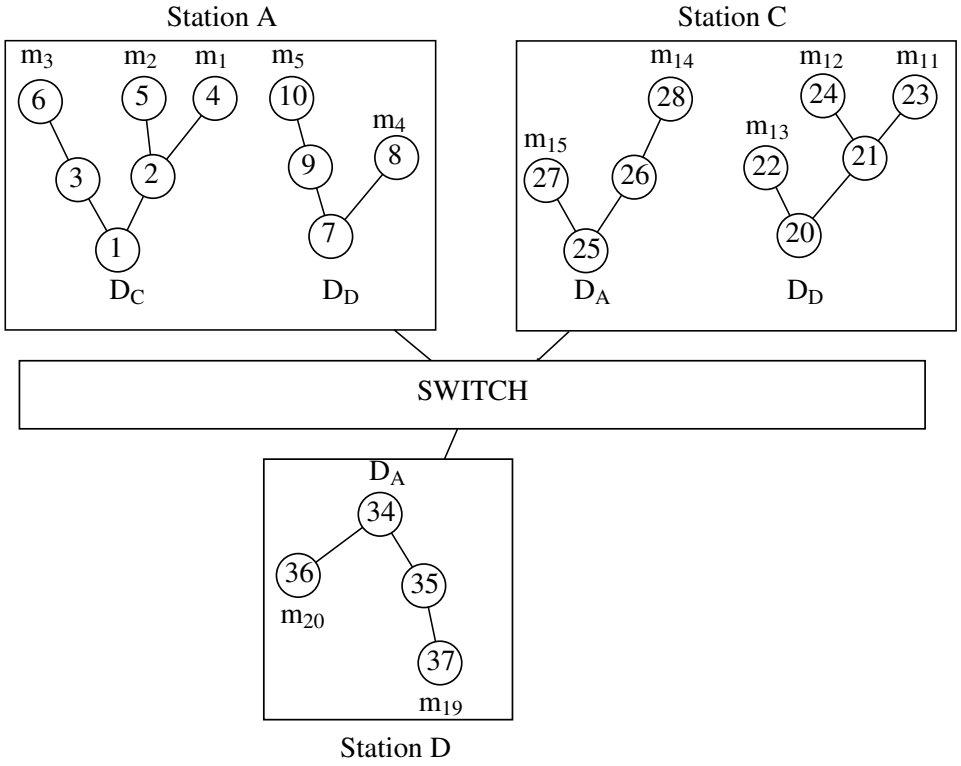


Figure 6.2: Independent Server Hierarchies (ISH) prepared for applications in each source station.

6.2.2 Experiments

In our experiments, we measure the response time of the message set when messages are activated periodically, i.e. every $Tmit$. We consider response time the duration in number of ECs measured between the EC in which a message request signal is received in the master node and the EC in which the message is dispatched by the master node ,i.e., included in the Trigger Message for transmission. Also, we verify that temporal isolation is achieved among message streams that share the network bandwidth. In particular, we consider three cases:

(a) temporal isolation between messages that belong to the same ISH thus the same application

(b) temporal isolation between messages that belong to different applications but share the same source node

(c) temporal isolation between message streams that belong to different ISHs but share the destination.

First, we study case (a) above and we consider an ISH U_A-D_C that contains message set $\{m_4, m_5\}$. To create a burstiness in the traffic generation by m_5, we activate it with an inter-arrival time that is shorter than 47 EC, which is its T_{mit}. We verify that the response time of m_4 is not affected by this change. Moreover, other messages in the network are not affected. Figure 6.3 shows the result for this case.

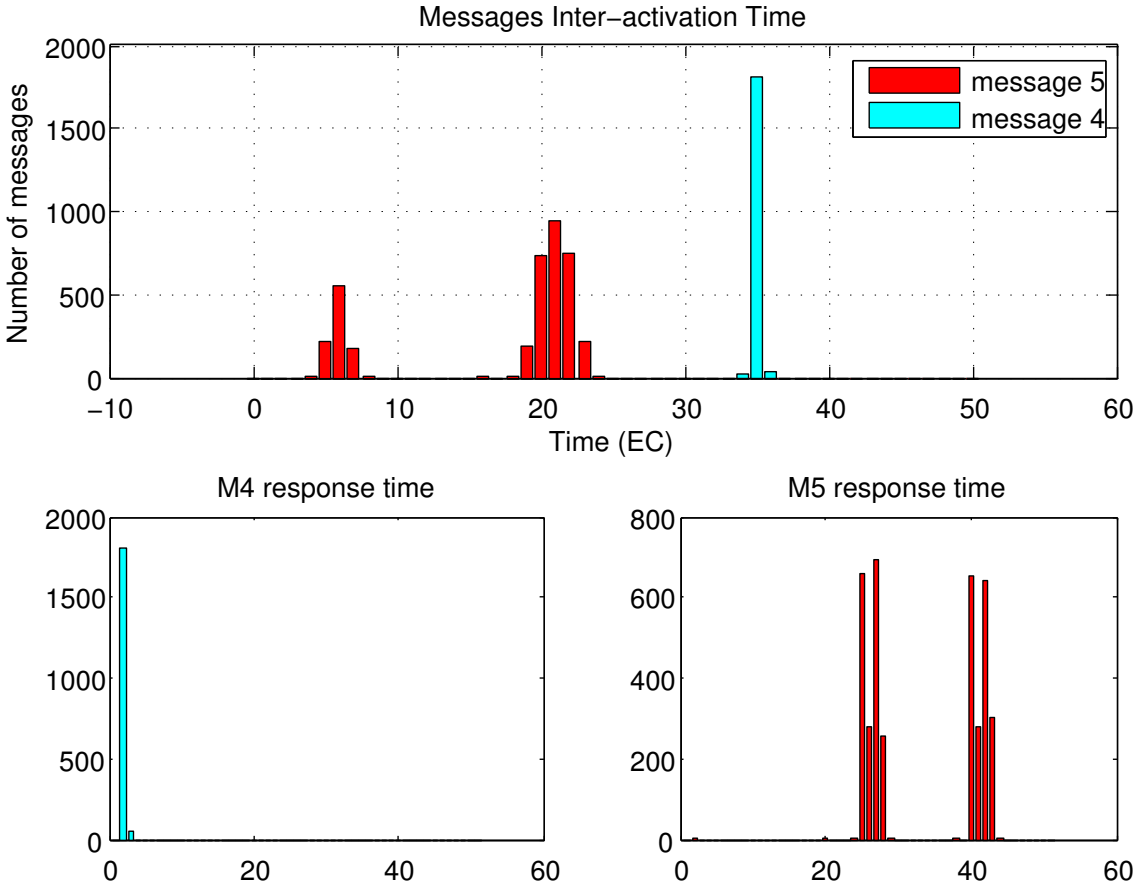
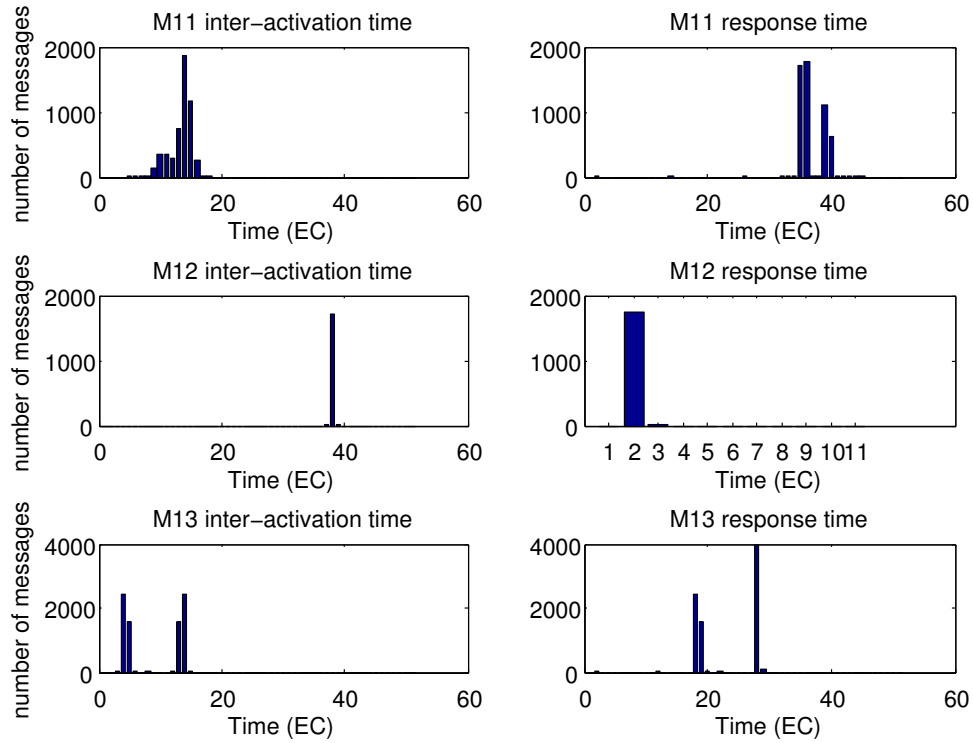


Figure 6.3: Message activations and response time for m_4 and m_5. m_5 has bursty activations

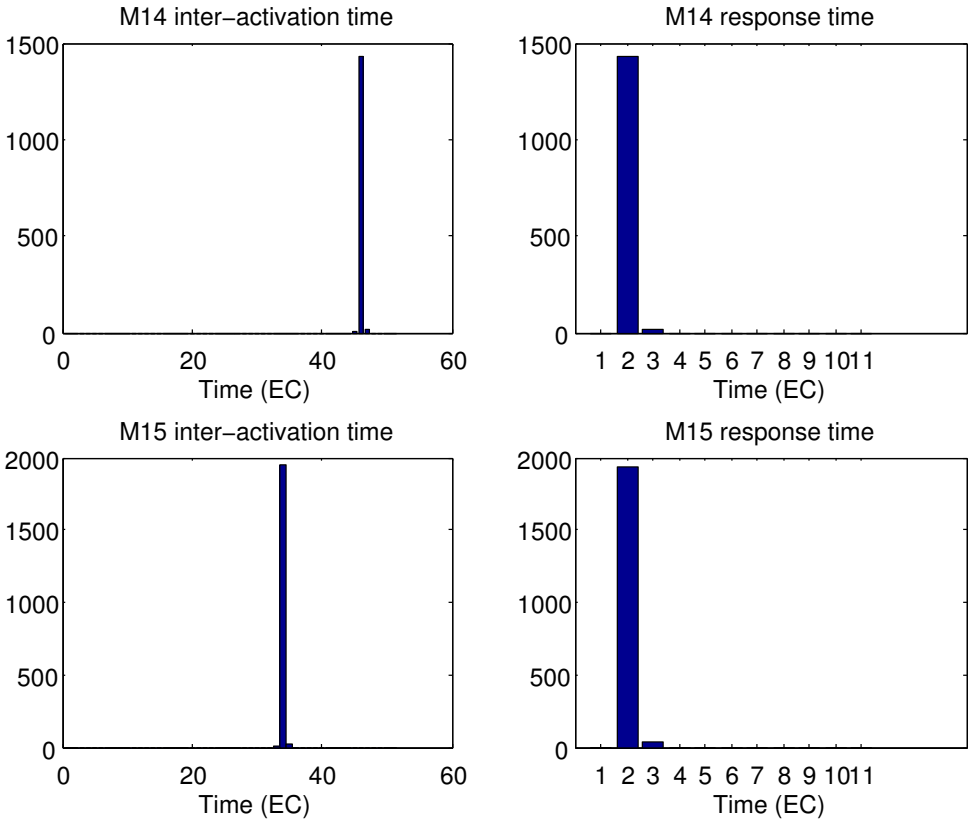
Next, we show the results of case (b) above; we consider station B that has two ISHs, i.e., U_B-D_C containing message set $\{m_{11}, m_{12}, m_{13}\}$ and another ISH U_B-D_A containing message set $\{m_{14}, m_{15}\}$. We make m_{11} and m_{13} bursty (generating them with inter-arrival times that are shorter then their own periodic activation) and verify that $\{m_{14}, m_{15}\}$ sharing the same source node are unaffected (Figure 6.4). Note that m_{13} has longer response times despite being in a separate branch of the hierarchy. The reason is that its server s_{22} has a period of 16 EC while the server s_{21} at the same level has a period of 8 EC. With RM policy s_{21} is favored over s_{22}. Also, s_{21} has a bursty stream connected to its child server s_{23}. This configuration has the effect of consuming most of the root server budget and resulting in longer response times for m_{13}. However, the periodic message m_{12} in the same ISH is unaffected. In particular, all the messages with periodic arrivals have a short response time between 1 and 2 EC. This is one order of magnitude improvement with respect to the polling HSF in Section 5.3.

6.3 Summary

This chapter addressed the implementation of hierarchical sporadic servers on Ethernet using FTT-SE. We presented the scheduling algorithm explaining in detail the specific replenishment management policy of hierarchical sporadic servers. Furthermore, we made an experimental validation of the system. Our results show that temporal isolation is achieved between message streams that are sharing the network bandwidth. Moreover, we report response times shorter than with the polling server policy.



(a) m_{11}, m_{12} and m_{13}



(b) m_{14} and m_{15}

Figure 6.4: Bursty activations in m_{11} and m_{13} do not impact the response time of other messages in the source station i.e. station B

Chapter 7

Design of Reservations

In previous chapters, we studied the feasibility of using network reservations with a Hierarchical Scheduling Framework (HSF) to support the design of complex systems. Reservation interfaces were manually chosen, potentially leading to inefficient resource utilisation. Moreover, when system size grows, choosing server interfaces that are adequate for system schedulability becomes a complex problem. This chapter presents an approach to design interfaces for reservations. We aim to build interfaces that are tight, i.e. provide the least capacity which is sufficient for a given workload schedulability. We focus on the design of polling servers in the hierarchies, applied to FTT-SE. We also validate our approach with extensive simulations using random message sets and hierarchies. Our approach to server design differs from other works in the literature on the same topic [110, 111, 112, 113, 143], because we consider non-preemptive packet scheduling and hierarchical polling servers. In particular, our approach focuses on meeting the following objectives:

(a) Guaranteeing that message deadlines are met.

(b) Generating tight server interfaces to ensure efficient use of the bandwidth (i.e., a minor decrease in the server capacity at any level of the hierarchy will create a deadline miss).

The chapter is organised as follows. Section 7.1 lists some considerations particular to the scheduling of hierarchial polling servers. This section also describes the system model and details the approach for designing server interfaces. Section 7.2 presents the experimental evaluation from our implementation. Finally, in Section 7.3 we give a summary of this chapter.

7.1 Server Design for Hierarchical Polling Servers

An important design consideration when designing a hierarchical set of servers is the use of the polling server policy (Figure 7.1). A polling server immediately discards its budget if it cannot schedule a child server. And, for a message to be served in an HSF, all the servers along the path from the leaf to the root server must be scheduled. For this reason, server periods along the path must be multiples of each other and in phase (condition 7.4). Otherwise, we may have a

situation in which messages receive bandwidth only at Least Common Multiple (LCM) of server periods on their path, which can typically lead to deadline misses, or they may not receive any bandwidth at all if the servers in the path are harmonic but are out of phase. Moreover, the constraint on simultaneous activation of all servers on any path of the hierarchy, typically leads to a strict periodicity requirement at the root level, i.e., without any jitter. This is a strong constraint for schedulability that, nevertheless, is typical in time-triggered communication schedules.

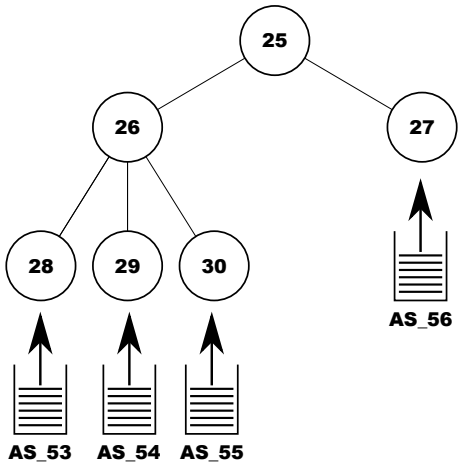


Figure 7.1: An example hierarchy with six servers $\{25 \cdots 30\}$) and four message streams $\{AS_{53} \cdots AS_{56}\}$

7.1.1 Modified system model

In what concerns the modelling of servers and message streams, we follow a slightly modified approach with respect to the model used in Chapters 5 and 6 as shown further on. A large message stream may generate several packets, which have a size of MTU (maximum transmission unit) bytes except for the last packet that can be smaller. The MTU value can be configured to determine the number of packets needed. Also note that, given the non-preemptive packets transmission, the MTU value determines the blocking that lower priority packets can cause to higher priority transmissions.

We consider N asynchronous streams (AS_x), modeled as sporadic according to expression 7.1, where C_x is the message transmission time of a stream AS_x that includes all overheads, $Tmit_x$ represents the respective minimum interarrival time and D_x the deadline, which we consider equal to $Tmit_x$. P_x identifies the parent server, i.e, the server to which the stream is connected to, and RT_x is its computed response time. For the convenience of notation, we define $\hat{C}_x$ as the corresponding message size in bytes, i.e., its original transmission requirement, $\tilde{C}_x$ as the corresponding message size in the number of packets and $\tilde{T}mit_x$ as the message minimum interarrival time expressed in FTT-SE elementary cycles (EC).

$$AS_x = (C_x, Tmit_x, P_x, RT_x, D_x) \tag{7.1}$$

A server Srv_x is characterized in expression 7.2 by its capacity C_x, replenishment period T_x, deadline D_x to provide its full capacity when requested. Moreover, the server Srv_x is associated with a parent server P_x and a corresponding computed upper bound on response time RT_x. Despite the similarity between the characterisation of servers and streams, there is a fundamental difference; streams would use the actual transmission time on the network, whereas servers merely characterize a reservation of the network resource. We add one more attribute, $List_x$, to the model of a server, which is the list of its child servers.

$$Srv_x = (C_x, T_x, List_x, P_x, RT_x, D_x) \tag{7.2}$$

Similarly to the streams model, we also define for convenience $\hat{C}_x$ as the corresponding server capacity in bytes, $\tilde{C}_x$ as the corresponding capacity in the number of packets of MTU size and $\tilde{T}_x$ as the replenishing period expressed in the number of EC. We are, thus, considering that the servers provide their capacity as integer multiples of the maximum size packet and with a period that is an integer multiple of the protocol EC.

7.1.2 Generating server interfaces

The presented approach has two steps. In the first step we generate interfaces for leaf servers and in the second step we generate intermediate and root server interfaces. The design approach aims to create server periods along the path that are multiples of each other and in phase. In the following, we formalize this condition.
Let us consider the stream AS_i which has the parent P_i (a leaf server in the hierarchy) and n is the number of levels in the hierarchy. Let $path_i$ denote the path in the server hierarchy that we use for scheduling AS_i. This path contains the leaf server where AS_i is connected and the parent servers upto the root of the hierarchy. Without loss of generality, we can refer root server to be at level 1 and the leaf servers at level n considering all branches of same depth (otherwise, consider any shorter branch extended with servers equal to its leaf until fulfilling the previous requirement). Then, to define $path_i$, we use an auxiliary element v which represents a parent server in the path, and $Par(x)$ returns the parent server for element v_x, i.e., $Par(v_x) = P_x$. Equation 7.3 gives the path used by AS_i whereas the condition for the multiplicity of periods along this path is formalized in 7.4.

$$
\begin{aligned}
v_0 &= P_i \\
v_k &= Srv_g \mid Srv_g = Par(v_{k-1}) \quad \forall k \in \{1 \cdots n-1\} \\
path_i &= \bigcup_{i=0}^{n-1} v_i
\end{aligned} \tag{7.3}
$$

Thus, the condition that we are interested in is given by the following expression:

$$
\forall v_x, v_y \in path_i \quad T_x = hT_y \ \wedge \ x < y \ \wedge \ h \in \mathbb{N} \tag{7.4}
$$

7.1.2.1 Interfaces for leaf servers

Given a message stream AS_x having the transmission requirement C_x and period $Tmit_x$, we describe an approach to generate candidate interfaces $(\tilde{C}, \tilde{T})$ to connect to its parent server P_x, where $\tilde{C}$ is the transmission capacity in MTU packets and $\tilde{T}$ is the replenishing period in number of elementary cycles.
Eq. 7.5 computes the number of packets that a parent server Srv_x shall handle per instance of a connected message stream.

$$
\tilde{C}_x = \left\lceil \frac{\hat{C}_x}{MTU} \right\rceil \quad (packets) \tag{7.5}
$$

Given that the stream AS_x will generate $\tilde{C}_x$ number of packets, at most, in a $Tmit_x$ interval, we can consider multiple interfaces $(\tilde{c}_{i,x}, \tilde{t}_{i,x})$ with $\tilde{c}_{i,x} \in \{1 \cdots \tilde{C}_x\}$, so that the original bandwidth requirement is respected (condition 7.6) where C_{MTU} is the time taken to transmit one MTU and LEC is the duration of the EC.

$$
\forall i \in \{1 \cdots \tilde{C}_x\}, \tilde{c}_{i,x}/\tilde{t}_{i,x} \times C_{MTU}/LEC >= C_x/Tmit_x \tag{7.6}
$$

To respect condition 7.6 the values for $\tilde{t}_{i,x}$ should be computed as in expression 7.7.
Note that all so generated interfaces in the list $Interface_x$ (Eq. 7.8) meet the original transmission request of AS_x and can be used for its parent server P_x. This procedure can be applied to all leaf servers in the hierarchy. The list of candidate interfaces $Interface_x$ may contain pairs having the same $\tilde{t}_{i,x}$ value for different $\tilde{c}_{i,x}$ values. For such a case, the presented approach keeps the pair with the minimum $\tilde{c}_{i,x}$ value and eliminates the remaining pairs.

$$
\tilde{t}_{i,x} = \left\lfloor \frac{\tilde{T}mit_x}{\left\lceil \frac{\tilde{C}_x}{i} \right\rceil} \right\rfloor \quad (EC) \tag{7.7}
$$

$$
Interface_x = \left\{ (\tilde{c}_{i,x}, \tilde{t}_{i,x}) : \ \forall i \in \{1 \cdots \tilde{C}_x\} \right\} \tag{7.8}
$$

7.1.2.2 Interfaces for intermediate servers

In the next step, we derive the interfaces for the intermediate and root servers of the hierarchy. This is done by composing the individual interface at each leaf server such that the corresponding parent interface can meet the combined demand of all the child servers.
Following our approach, for each intermediate server in the hierarchy we explore the space of $Interface_k$ of all its users (child servers) computing the greatest common divisor (GCD) of periods for each composition of child interfaces.
We then pick the child interfaces whose periods produce the maximum GCD. The maximum GCD value computed above is taken as the period of the parent server being designed and its capacity is the sum of the capacities of the corresponding child interfaces.
To develop our approach, we define the lists $\mathbb{T}_k$ and $\mathbb{C}_k$ as given in expression 7.9 and 7.10. These lists contain, respectively, the $\tilde{t}_{i,k}$ and $\tilde{c}_{i,k}$ values of all the interface candidates in $Interface_k$.

$$\mathbb{T}_k = \{\tilde{t}_{i,k} : \tilde{t}_{i,k} \in Interface_k \forall i \in \{1 \cdots \tilde{C}_k\}\} \tag{7.9}$$

$$\mathbb{C}_k = \{\tilde{c}_{i,k} : \tilde{c}_{i,k} \in Interface_k \forall i \in \{1 \cdots \tilde{C}_k\}\} \tag{7.10}$$

Then, for Srv_x we define S_x (Eq. 7.11) as the space of all possible combinations of the interfaces of the respective N_x child servers (note that N_x is the cardinality of $List_x$).

$$S_x = \{(\mathbb{C}_1, \mathbb{T}_1) \times \cdots \times (\mathbb{C}_{N_x}, \mathbb{T}_{N_x})\} \tag{7.11}$$

Note that, when the child servers are leaf servers, the cardinality of S_x is upper bounded by $\prod_{k=1}^{N_x} \tilde{C}_k$. However, for higher levels in which there is no leaf server in $List_x$, then S_x is restricted to a single combination.
Once we have defined S_x we now define the set $\mathbb{D}_x$ (Eq. 7.12) which contains the GCD of the periods of all combinations in S_x. Note that both $\boldsymbol{c}$ and $\boldsymbol{t}$ are vectors with N_x dimensions.

$$\mathbb{D}_x = \{GCD(\boldsymbol{t}) \ \forall (\boldsymbol{c}, \boldsymbol{t}) \in S_x\} \tag{7.12}$$

Finally, if (c_v, t_v) is the point in S_x that corresponds to the maximum element in D_x, then the interface for an intermediate or root server I_x can be defined as in Eq. 7.13.

$$I_x = (Sum(c_v), GCD(t_v)) \wedge (c_v, t_v) : GCD(t_v) = max(\mathbb{D}_x) \tag{7.13}$$

7.1.2.3 Illustrative example

We consider an example to understand the presented approach. Consider a scenario as shown in Fig. 7.1. This is an example hierarchy that has six servers $\{Srv_{25}, \cdots, Srv_{30}\}$ and four message

streams $\{AS_{53}, \cdots, AS_{56}\}$. Table 7.1 lists the parameters for the given message set. The size of MTU is $\hat{C}_{MTU} = 1492$B or $C_{MTU} = 128$ μs at 100 Mbit/s. Note that the transmission times already account for all packet transmission overheads.

Using Equations 7.5 – 7.8, we generate candidate interfaces at the leaf servers where mes-

Table 7.1: Message Parameters

Id(x)	$\tilde{T}mit_x$ (EC)	$\hat{C}_x$ (B)	C_x μs	$\tilde{C}_x$
AS_{53}	38	7065	613	5
AS_{54}	25	3393	300	3
AS_{55}	54	6341	555	5
AS_{56}	60	2715	236	2

sage streams are connected. Let us consider the case of AS_{53}; using Eq. 7.5, first we find it has $\lceil \frac{7065}{1492} \rceil = 5$ packets. Then, we find what interface shall Srv_{28} have considering that it schedules from 1 to 5 packets at each scheduling instant. We explain these cases in turn:

If Srv_{28} serves a single packet ($\tilde{c}_{1,53}$) then, $\tilde{t}_{1,53} = \left\lfloor \frac{38}{\lceil \frac{5}{1} \rceil} \right\rfloor = 7$ and hence the resulting interface $(1,7)$ can fulfil the AS_{53} demand;

if Srv_{28} serves two packets ($\tilde{c}_{2,53}$) then, $\tilde{t}_{2,53} = \left\lfloor \frac{38}{\lceil \frac{5}{2} \rceil} \right\rfloor = 12$, interface is $(2,12)$, and so on (Figure 7.2). We can see in Figure 7.2 (top) an instance of AS_{53} arriving at $t = 0$ and its deadline is at $t = 38$. Then, we have a timeline for five different interfaces for Srv_{28}. While, all these interfaces can fulfill the transmission request, there is a trade-off between server utilization and message response time. Also, the leaf servers are composed at the intermediate and root levels. And, the interface at leaf is chosen in such a way that combined resource requirement at the root level is optimized.

Table 7.2 lists the interface candidates for the leaf servers.

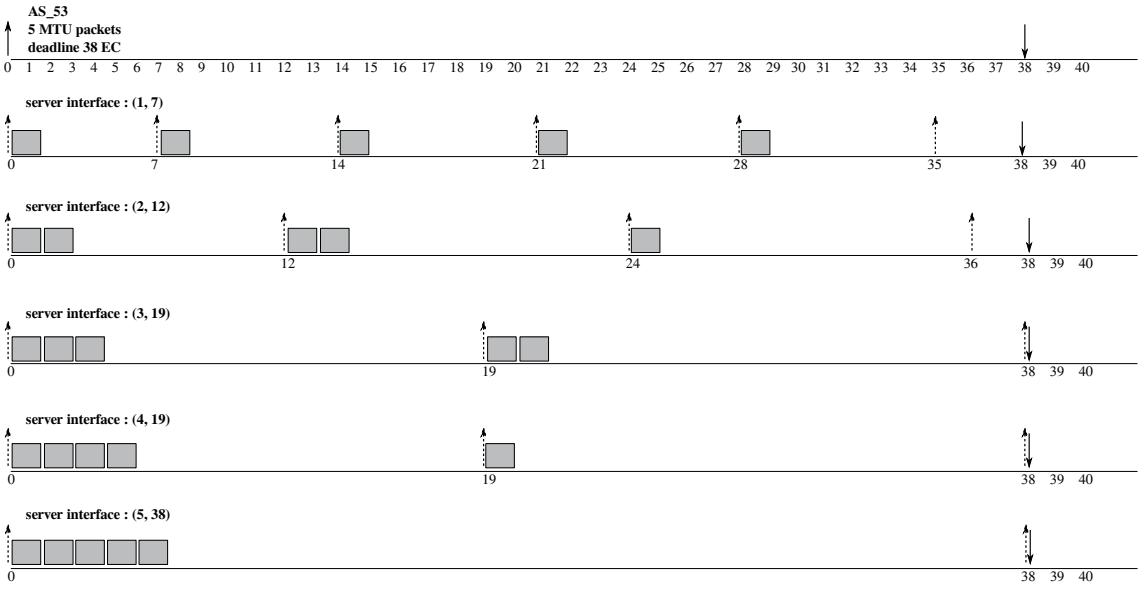


Figure 7.2: Different candidate interfaces for Srv_{28} to schedule AS_{53}

In the next step, we derive the interface for the intermediate server i.e., Srv_{26} as well as choose an

Table 7.2: Interface candidates for leaf servers

Id(x)	$Interface_x$ (packets, # EC)
Srv_{28}	(1, 7), (2, 12), (3, 19), (4, 19), (5, 38)
Srv_{29}	(1, 8), (2, 12), (3, 25)
Srv_{30}	(1, 10), (2, 18), (3, 27), (4, 27), (5, 54)
Srv_{27}	(1, 30), (2, 60)

interface for the leaf servers from their respective candidate interfaces. Let us consider the server Srv_{26} with $List_{26} = \{Srv_{28}, Srv_{29}, Srv_{30}\}$. Using expression 7.12 and Eq. 7.13, we choose the combination that gives the maximum GCD. In this example, we obtain $(2,12)$ for Srv_{28}, $(2,12)$ for Srv_{29} and $(2,18)$ for Srv_{30} meaning 2 MTU packets every 12 or 18 EC, which maximizes the GCD among all period combinations, giving the gcd value of 6 EC.

And, then using Eq. 7.13, the interface at Srv_{28} is taken as $(2+2+2, gcd(12,12,18))$ or $(6,6)$ i.e., 6 MTU packets every 6 EC. The leaf Srv_{27} have no sibling servers, hence, we choose interface $(1,30)$ at Srv_{27}.

Finally for the root server, we compose interfaces from Srv_{26} and Srv_{27}. At this step, we sum up the capacities from child interfaces and the period is chosen as the GCD of the periods from each child. Thus the interface for the root server of the hierarchy Srv_{25} is $(6+1, GCD(6,30))$ or $(7,6)$ which translates into 7 MTU packets as the server capacity and 6 EC as its period. We refer to the design approach as *rational approach*. Figure 7.3 depicts the interface composition for the presented example using this approach.

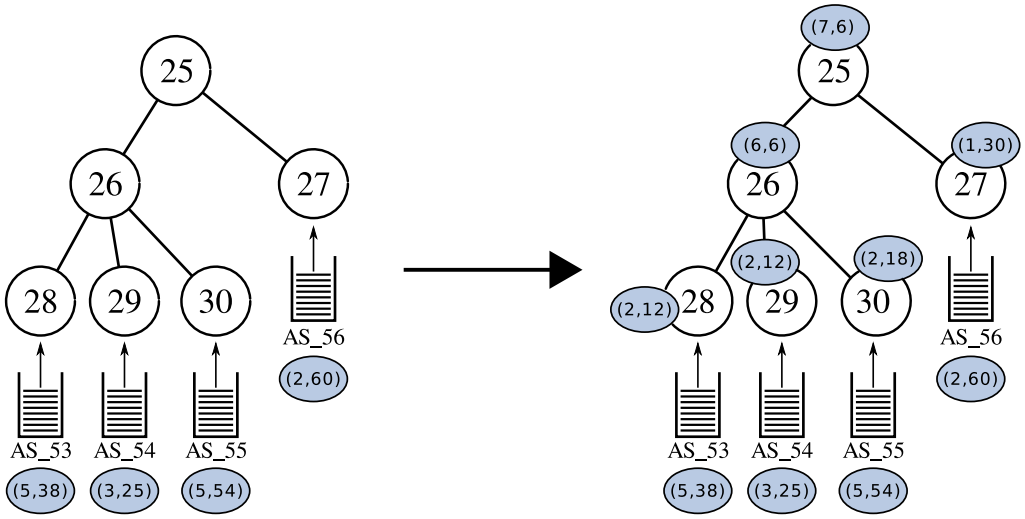


Figure 7.3: Interface composition with rational approach

7.1.2.4 Schedulability issues

In what concerns schedulability, two base assumptions are fundamental to this work, namely the strict periodicity of the root servers and the periods of the servers along each branch of the hierarchy being integer multiples of each other.
Starting from the leaf servers, the way we construct the interfaces, under strict periodicity, assures that every message stream AS_x will always get at least C_x service within $Tmit_x$, thus allowing it to meet its deadline.
Then, when we compose several intermediate servers, the composition rules we follow, together with the strict periodicity constraint, also guarantee that any user, Srv_x or AS_x, receives a service at least equal to C_x within its $Tmit_x$. Note that $Tmit_x$ is an integer multiple of the intermediate server period and provides, in each activation, service equal to the sum of all capacities or execution requirements of its users.
Therefore, our method assures schedulability as long as total utilization of the root servers is less than 1.

7.2 Evaluation

We aim to verify that the message sets are schedulable with the presented approach and a large percentage of messages is scheduled close to the respective deadlines. This indicates that the approach assigns tight capacities to the servers.

We test our approach with simulation and randomly generated message sets and hierarchies. The simulator executes the actual traffic scheduling EC by EC, exactly as the real operation of the protocol and thus we consider it accurate. Message activation pattern is sporadic where we enforce at least $Tmit$ separation between consecutive activations. The specific activation instants are chosen randomly with uniform distribution between 0 and $\tilde{T}mit - 1$ after the enforced $Tmit$ separation to the previous activation. The duration of EC is 20 ms, and length of the asynchronous window is 90% of the EC. Message period is chosen between 10 and 65 EC, and duration between 128μs to 896μs, i.e., 1 and 7 MTU packets. Deadline for each message is equal to its period. We run our simulation with a large dataset comprising 1198 message sets where each message set can comprise up to 72 messages depending on the configuration (hierarchies) generated for a particular simulation run. We then observe the message response times.

7.2.1 The worst-case behavior

In this experiment, we measure the minimum time to deadline upon completion of message transmission. It shows how close the response times are to the message deadlines. Let us consider a message set M with n message streams. For each message stream $AS_i \in M$ $(1 \leq i \leq n)$, first, we log the time to the deadline for each of its scheduling instances, and we keep the minimum in min_ttd_i. Then we save all observed minimum values for all n variables in a list $TTDmin$. Finally, we take the minimum time to the deadline for M, i.e., $min(TTDmin)$. With each message

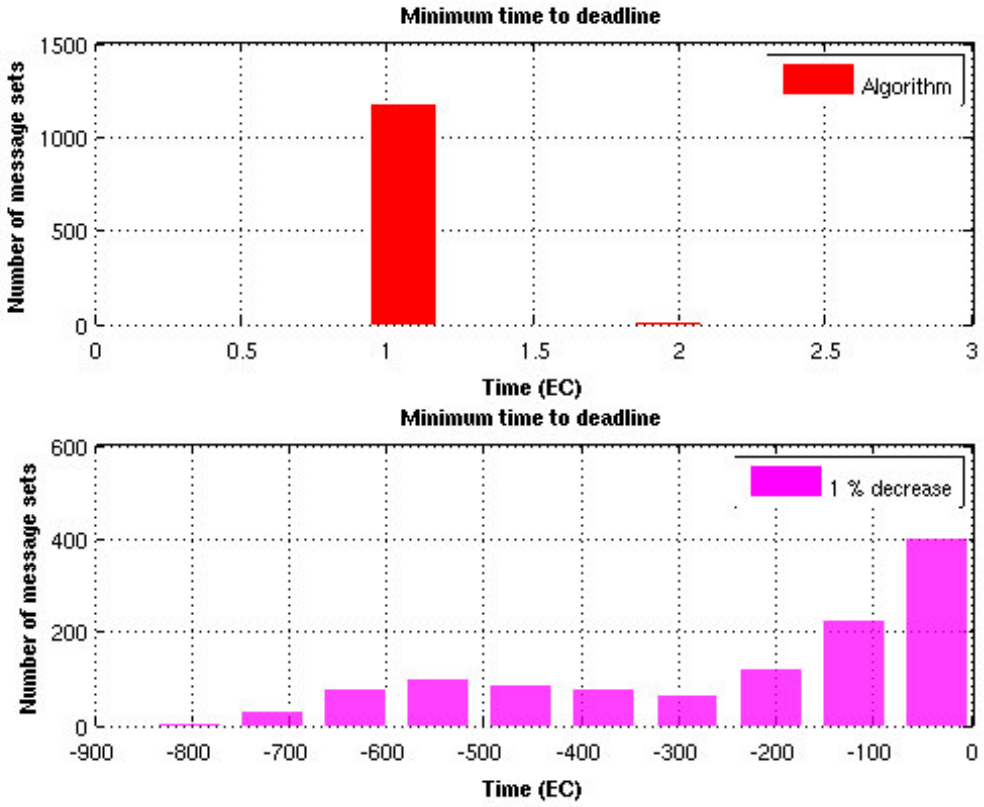


Figure 7.4: Minimum time to the deadline for the message sets

set, first we check its schedulability; next, we decrease the capacities assigned to all servers by 1% and rerun the simulation while keeping the same configuration (i.e. hierarchies and message sets). Fig. 7.4 shows the results. We can see on top that with the proposed design approach 98% of the message sets have at least one message with a response time 1 EC away from the deadline, and with a 1% decrease in server capacities we observe a system saturation, with big deadline misses. This is an empirical confirmation that our approach generates tight interfaces.

7.2.2 The average case behavior

Since each message in the set may have a different period, we normalize the response time value for a particular message with respect to its period as follows:

$nrt_x = (RT_x/Tmit_x) * 100$

where RT_x is the mean response time among all instances of the message stream AS_x. The value nrt_x is saved in the list $NRTmean$ that contains the average values for all messages in the respective set. Then, we compute the average normalised response time for the message set as $mean(NRTmean)$, and we plot the histogram of these values for all message sets in Fig. 7.6. Larger values indicate messages are scheduled closer to the respective deadlines, thus with less slack indicating a tighter schedule. We can see that for 90% of the message sets the average

response times are above 70% of the respective deadlines. To present the effectiveness of this approach, we also compare the average results with those obtained with a *naive approach*. In this approach, leaf server's interface is generated directly from the connected message stream model, i.e., $(C_x, Tmit_x)$. Then for the intermediate/root servers, the interface capacity is taken as the sum of child interface capacities and the period is the gcd of child interface periods. Figure 7.5 shows the interface composition using the *naive approach* for the example ISH given in Section 7.1.2.3. Using the *naive approach*, we see that message sets have average response times no larger than 50% of the respective deadlines.

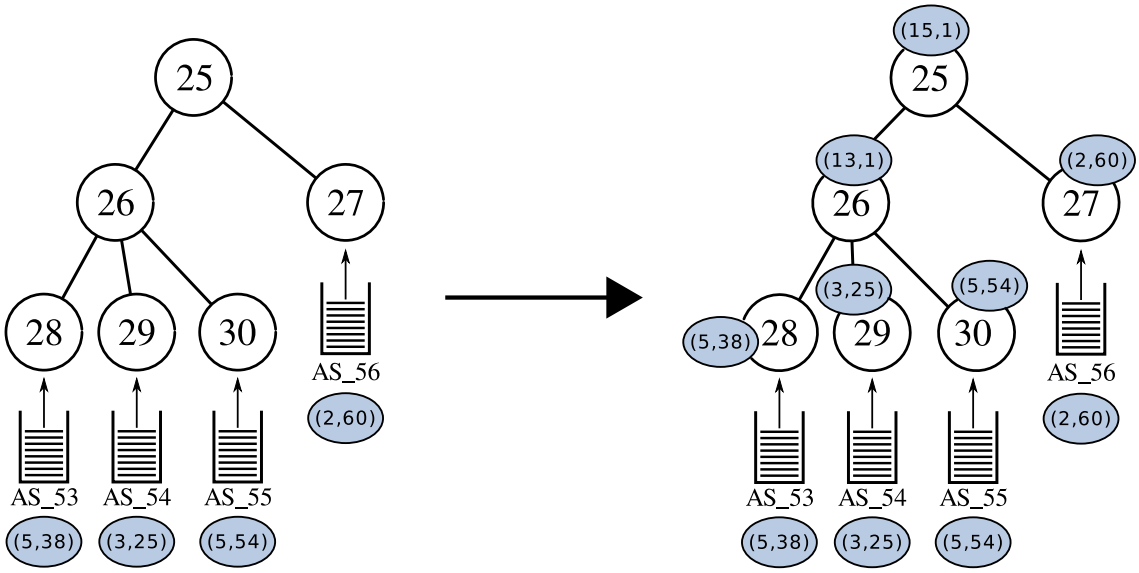


Figure 7.5: Interface composition with naive approach

7.2.3 Root server utilization

The interface at root server in a hierarchy abstracts the combined resource requirement from all the connected streams in that hierarchy. For each simulation run, the message set is scheduled with a set of hierarchies (1 or more). We compute the total root server utilization for all the hierarchies. If this value goes beyond 100%, then the approach aborts. We plot the histograms for bandwidth taken by the root servers for all message sets in Fig. 7.7. We can see that , on average, the proposed approach leads to root servers that require less than half of the bandwidth requested by the naive approach. This result also explains the difference in average normalized response time, and it is consistent with the observed tightness of the server interfaces designed with the proposed method. Therefore, the proposed method leads to a substantially more efficient design than a naive approach as described. The reason is the better packing of servers that the proposed method offers.

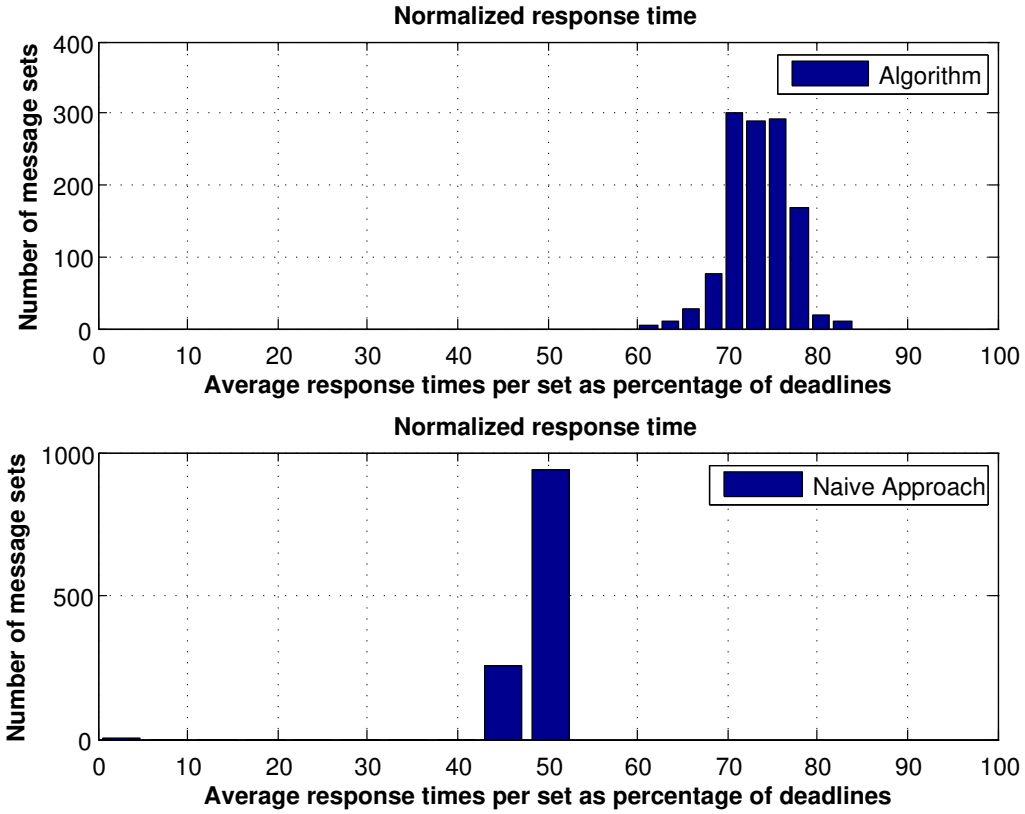


Figure 7.6: Distribution of average response time as percentage of deadline over all message sets

7.3 Summary

In this work, we presented an approach for server design in a multi-level hierarchical server-based architecture over Ethernet with FTT-SE. We considered the polling server policy and the non-preemptive nature of packet scheduling. Our results indicate that message sets are schedulable with the proposed interfaces, and in general message response times are close to the deadlines. Given an independently generated workload, we observed that the approach assigned tight server interfaces, i.e., with low slack. In fact, a decrease in server capacities as small as 1% saturates the system leading to strong deadline misses. This points in the direction of an HSF design that requests the least system resources.

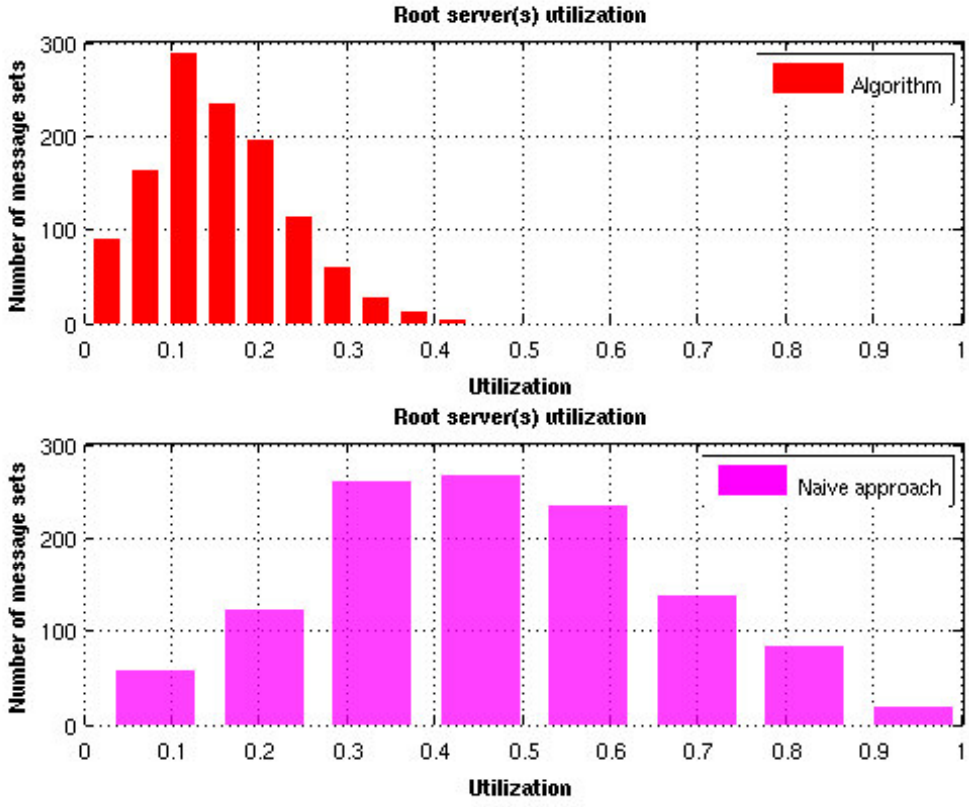


Figure 7.7: Total utilization of the root servers per message set

Chapter 8

Experimenting with Hierarchical Reservations on FTT-SE

In this chapter, we present the experimental framework that we used for generating large, random test data sets and server hierarchies. Such data is used for verification of hierarchical server-based scheduling strategies. From the implementation point-of-view, FTT-SE has a layered structure with three layers namely Interface, Management and Core layer. The protocol basic communication mechanisms such as transmission control and traffic scheduling are implemented within the core layer [137]. In our experiments, we used a simplified structure that bypasses the first two protocol layers and interacts directly with the core layer. Figure 8.1 shows a layered view of the protocol structure and also depicts the level at which the simulator works. With such setting, a sample test configuration can be used simultaneously to evaluate the traffic scheduling within the FTT-SE core layer and analyse the corresponding scheduling strategy. This helps to verify the system timing. In this chapter, we describe design principles and implementation aspects of the simulator, and at the end, we discuss the design flow of applications using our HSF and how they can be deployed.

8.1 Components of the Experimental Framework

For our experimental framework, focused on traffic scheduling with multi-level server hierarchies, we defined the following essential capabilities as requirements:

(a) Specifying the structure of hierarchies

(b) Generating application messages

(c) Filling in the hierarchies with server parameters

(d) Generating a repository of server messages

(e) Processing hierarchies (message scheduling) upon events such as message activations and server replenishments

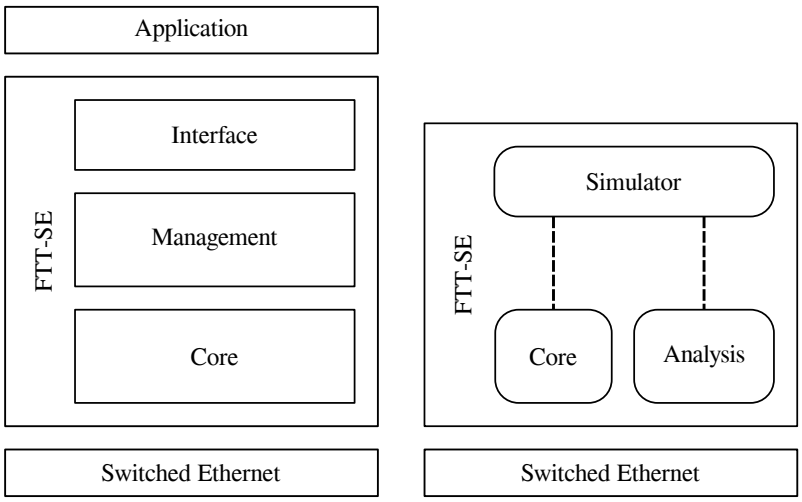


Figure 8.1: FTT-SE internal layering (left) and experimental platform (right)

8.1.1 Structure of hierarchies

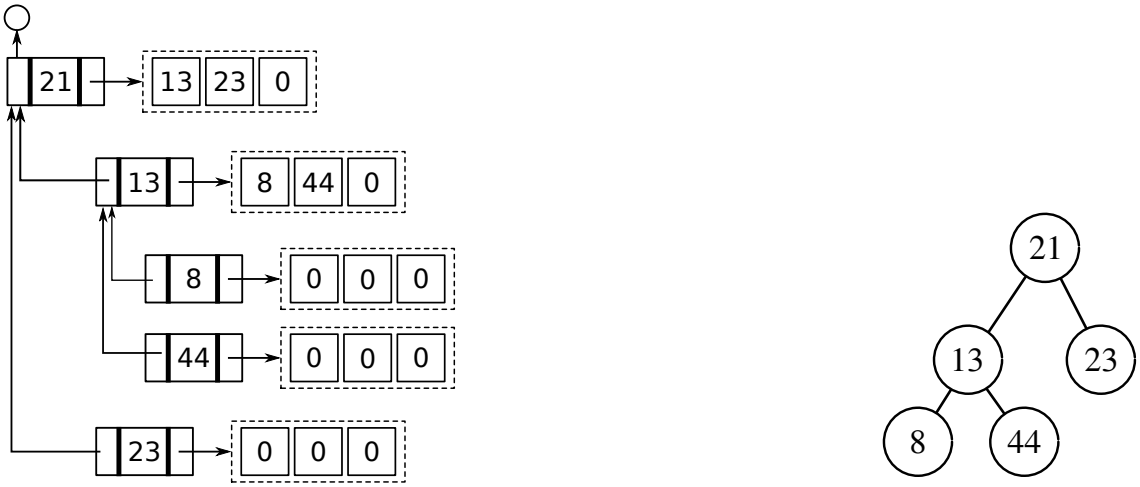
Multi-level hierarchies exist in the form of a tree structure which we refer to as an *Independent Server Hierarchy* (ISH). Within an ISH, the bandwidth allocated to the root server can be partitioned at different levels, partitions being allocated to different logical sub-groups. We manage the communication of an application with one ISH. Then, different streams in that application can be assigned to different parts of the ISH. It becomes explicit that the application and hence the individual streams therein have the same communication endpoints, i.e., source and destination. The ISH representation and corresponding implementation follows the concept of *adjacency list* which is commonly used to represent directed graphs. The code listing (see Listing 8.1) shows a partial view of the structure of an ISH. Notice the presence of two I_Ptr fields namely list and parent; any node in the hierarchy can have several child nodes and i^{th} child is indexed in list[i]; a reference to the parent of any node in the hierarchy is given by a parent. I_Ptr is a pointer of the type ish_node. In other fields item is the unique id of a given node, num_child_nodes is the number of child servers and type defines if it is a ROOT, INTERMEDIATE or a LEAF server.

Listing 8.1: The structure representing an ISH node

```
struct ish_node
{
        int item;
    I_Ptr list[MAX_COLS-1];
    I_Ptr parent;
    int num_child_nodes;
    ISH_NODE_TYPE type;
}
```

An adjacency list representation for a graph associates each vertex in the graph with the collection of its neighbouring vertices. For example, Figure 8.2 shows an adjacency list representation (right) for an example ISH with 5 servers (left). Notice the pointer to the parent and the list of

child nodes. MAX_COLS given in Listing 8.1 is defined as 4 and hence we assume a maximum of 3 child nodes per server (MAX_COLS − 1). The hierarchical structures are randomly generated with the possibility that no child server is generated which we indicate by a 0 in the adjacency table, and this will be a null child in the ISH. The leaf nodes have all null children, represented by 0 entry in the corresponding adjacency tables, e.g., nodes 8, 44 and 23).



(a) Adjacency list (b) Corresponding ISH

Figure 8.2: Adjacency list represenation for an ISH

8.1.2 Generating application message set

In our system, there may exist messages that are scheduled with servers, and other messages that are scheduled directly through the system level scheduler (refer Section 5.1.3). In this case, the application message set is generated differently, the difference being that server-based traffic needs mapping to a server in the system. We describe here the case of server-based traffic. We generate application messages and associate these to servers in our system. Messages are associated with leaf servers only. Given the list of available leaf servers, we associate one message per leaf. The total size of the message set thus equals the number of leaf servers in the system. In this work, we make no preference concerning the ISH, or server in that ISH, to which each message is mapped. The algorithm to generate a message set is shown in Algorithm 5. This algorithm assumes that hierarchies (structures) have been prepared and are stored in the ISH database, which yet contains only the skeleton, i.e., structures with ids of the servers already specified. The algorithm picks next ISH from the database (element in line 5), and retrieves the list of its leaf servers (leaf_set in line 6). Then, it generates an application message corresponding to each leaf server in the ISH, assigning attributes such as period, deadline etc. (lines 8-15). The message inherits the communication endpoints, i.e., source and destination, from the connected server (line 15).

Algorithm 5 Generating a message set, server-based traffic

```
1:  function MSG_prepareSet()
2:      msg_id ← get_max_id_used() + 2          ▷ generate a unique message id
3:      j ← 0
4:      while j < ISH_db_getSize() do
5:          element = ISH_db_getElementbyIndex(j)
6:          leaf_set ← element.leaf_array
7:          for each item u in leaf_set do
8:              msg_id ← msg_id + 1
9:              set_maximum_id(msg_id)
10:             msg_size ← random(MTU, 7 × MTU)
11:             msg_period ← random(10, 70)
12:             msg_deadline ← msg_period
13:             msg_producer ← element.source
14:             msg_consumer ← element.destination
15:             msg_server ← u
16:         j ← j + 1
```

8.1.3 Filling in server parameters

We can choose the interfaces for servers in a multi-level hierarchical structure according to three approaches with different complexity. These approaches are so-called the *random* composition, the *rational* composition, and the *naive* composition approach. We have described the latter two methods in Chapter 7. The random composition approach is based on heuristics and will not always generate efficient interfaces or schedules without deadline misses. In our system, the message size comprises a specified number of packets (a packet is the size of an MTU) chosen randomly within a given range (for instance, between 1 and 7 MTU packets). Also, since only one stream is connected per leaf server, and at run-time, we do not know which is the size of stream (owing to the random selection), then, we can either choose an interface at the leaf server with a capacity larger than the maximum stream size (for instance, over 7 MTU packets) and may potentially incur wasted bandwidth when the size of associated stream is smaller, or we may choose random capacity of the server in the same range (the one used to generate the stream), and incur missed deadlines when the corresponding stream size is larger. For small systems, however, we can set server parameters such that the capacity of the server is enough to schedule an instance of the stream, and follow simple composition rules, e.g., the capacity of the parent servers is more than the sum of their child servers. However, this approach is hard to accomplish when we have large systems with several applications.

8.1.4 Repository of servers

The servers repository is the database of existing servers in the system. For any server in the system, there is a corresponding entry in the repository that maintains a view of that server static and

dynamic attributes. Static attributes include period, budget, id, level in the hierarchy and communication end-points, whereas dynamic information includes its current capacity, finish status, state and overrun flag. Each server with its attributes that include the interface created in the previous step is stored in the repository, which is used by the system scheduler during system execution.

8.1.5 Message activations

In our system, message activations can be periodic or sporadic. Additionally, we can induce bursts in a randomly chosen subset of the messages. The algorithm to activate the message set is shown in Algorithm 6. Each simulation is run for a configurable number of cyles, denoted simulation_cycles. Other input parameters include the message set MSG, its size sz and the activation option indicating whether the messages will be activated periodically or sporadically given respectively with PERIODIC or SPORADIC. The algorithm to generate periodic activations is simple; given the total number of simulation cycle and message k period, we find the total activation instants (line 3) for message k. Then, the array ACT[k] will contain the actual activation instants for MSG[k] (line 8). With sporadic activations, the inter-arrival time between successive instances of a message is chosen randomly in the interval (T_k, $2T_k - 1$) where T_k is the message k period (lines 9 - 11).

The principle of the algorithm is to prepare arrays with corresponding activation instants for each message and carry out message activations at specified instants during the system run. In particular, during each cycle, we increment a counter h. For each message, we perform a binary search in its activation array using the C library function **bsearch**. This function searches an array of NUM_ACT[k] objects, the initial member of which is pointed to by the base ACT[k], for a member that matches the object pointed to, by h. The size of each member of the array is specified by function second last argument, whereas the last argument is the function to compare two elements. When there is a matching activation in the cycle h, the message is activated (lines 13 - 17). After the pending activations for the current cycle have been updated, the scheduler is invoked.

To account for the messages with bursty activations, we have formulated a method to create bursts in randomly chosen messages, denoted by num_bursty in Algorithm 7. An important component of the algorithm is the array seq which contains the indices of the messages in MSG. We randomise the contents of this array (line 5), and then we choose the first num_bursty elements from this array. These are the indices of the messages from MSG that we shall induce bursts in. The routine check_if_in_seq checks for a message at index k if it belongs in the first num_bursty elements of seq (lines 7 and 10). For a bursty message, the method involves increasing the size of NUM_ACT array. In our settings, we increase the size 3 times or 200%. Next, for preparing the ACT array, we prepare the successive activations instants which are chosen randomly such that separated by $T_k/2$ or smaller (line 15). To reflect a different bursty pattern, we can change the parameter num_bursty or the size of the array NUM_ACT in line 8.

Algorithm 7 is used in combination with Algorithm 6 where an additional flag in Algorithm 6 can control whether the bursty behavior is simulated.

Algorithm 6 Message activations

```
Input: MSG, sz, act_option
1:  procedure activate_message_set                          ▷ activating the message set
2:      while k < sz do                                     } prepare activation instants for each message, periodic or sporadic
3:          NUM_ACT[k] = simulation_cycles/MSG[k].msg_period
4:          k ← k + 1
5:      while k < sz do
6:          r ← 0
7:          for g in NUM_ACT[k] do
8:              ACT[k][g] ← g × MSG[k].msg_period
9:              if act_option = SPORADIC then
10:                 r ← r + random(0, MSG[k].msg_period − 1)
11:             ACT[k][g] ← ACT[k][g] + r
12:     h ← 0
13:     while h < simulation_cycles do
14:         for k in 0 : sz − 1 do
15:             item = bsearch(&h, ACT[k], NUM_ACT[k], sizeof(int), cmpfunc)
16:             if item != NULL then
17:                 MDB_update_status(MSG[k].msg_id)
18:         invoke_scheduler()
19:         h ← h + 1
```

Algorithm 7 Formulating the bursty activations

```
Input: num_bursty, sz
1:  procedure induce_burst                                  ▷ inducing bursty behavior
2:      while m < sz do
3:          seq[m] ← m
4:          m ← m + 1
5:      randomize_seq(seq, sz)
6:      for k in 0 : sz − 1 do
7:          if check_if_in_seq(k, seq, num_bursty) then
8:              NUM_ACT[k] ← (NUM_ACT[k] × 200)/100
9:      for k in 0 : sz − 1 do
10:         if check_if_in_seq(k, seq, num_bursty) then
11:             f ← 0
12:             ACT[k][0] ← MSG[k].msg_period
13:             for g in 1 : NUM_ACT[k] − 1 do
14:                 f ← random(0, MSG[k].msg_period/2)
15:                 ACT[k][g] ← ACT[k][g − 1] + f
```

Following the message activations, the system scheduler is invoked; it inspects the system databases and schedules all the traffic (server-related and legacy traffic) in each cycle (Algorithm 6, line 18). The system scheduler will process the hierarchies according to the algorithms detailed in Chapter 5, and Chapter 6.

8.2 Application Design and Execution Flow

For a random simulation of our system, we specify two important arguments to the simulator, namely, the maximum number of slave stations in the network, and the maximum number of ISHs on each slave. These two arguments can determine the maximum number of messages in the system. For instance, in a particular simulation, where we choose 3 slave stations and 3 ISHs per slave, we can have a maximum of 9 ISHs. Then, considering other implementation specific configurations (i.e., maximum of 3 levels in the hierarchy and a maximum of 3 child nodes for a given parent server node), we can have 9 messages in one ISH, and hence, for this example configuration, we will have 81 messages in total. Figure 8.3 shows how our framework is used.
Here, we assume that the two main arguments (number of slaves and ISHs) have been passed to the program. The figure depicts the design and execution flow divided into two main phases, the system setup phase and the system execution phase. By system execution, we refer to the schedule building by the FTT-SE master in successive elementary cycles.
Notice the separation between the application program and the FTT-SE core layer. The application program builds the main components of the system such as message sets, hierarchies and servers. Moreover, the server design logic is contained on the application side. However, an HSF demands to build two additional databases within FTT-SE core. These databases are the server repository (Section 8.1.4) which stores information regarding all the servers in the system, and the server hierarchy database. The latter is instrumental in establishing relations between different servers in the system. For instance, we consult the hierarchy database to ask such questions as, which hierarchy does the server belong to, or which are the child nodes or the parent node of a given server. During the system setup, the application program can create and update these databases within the core through specific calls as shown in Figure 8.3. During the system execution, messages are activated, and scheduler is invoked. The scheduler enforces the principles of hierarchical server-based scheduling concerning capacity consumption, server replenishment management, and prepares transmission schedules.
For experimentation purposes, sometimes, it is necessary to rerun the system by changing some parts of the configuration while keeping other parts of the configuration constant. For example, in server design approach (refer Chapter 7), initially, we run the system with sample hierarchies. Next, we keep the same message set and hierarchies but change the server capacities by a certain value, and we rerun the system. Our purpose is to understand the impact on the application response time. This exercise requires storing the system configuration and then using it for subsequent runs of the system. This is challenging when our system consists of several input data that

are required for its functioning. Our simulator supports this functionality. We can pass specific arguments to the simulator, so that, it can either run a fresh simulation, or the one based on previous input data.

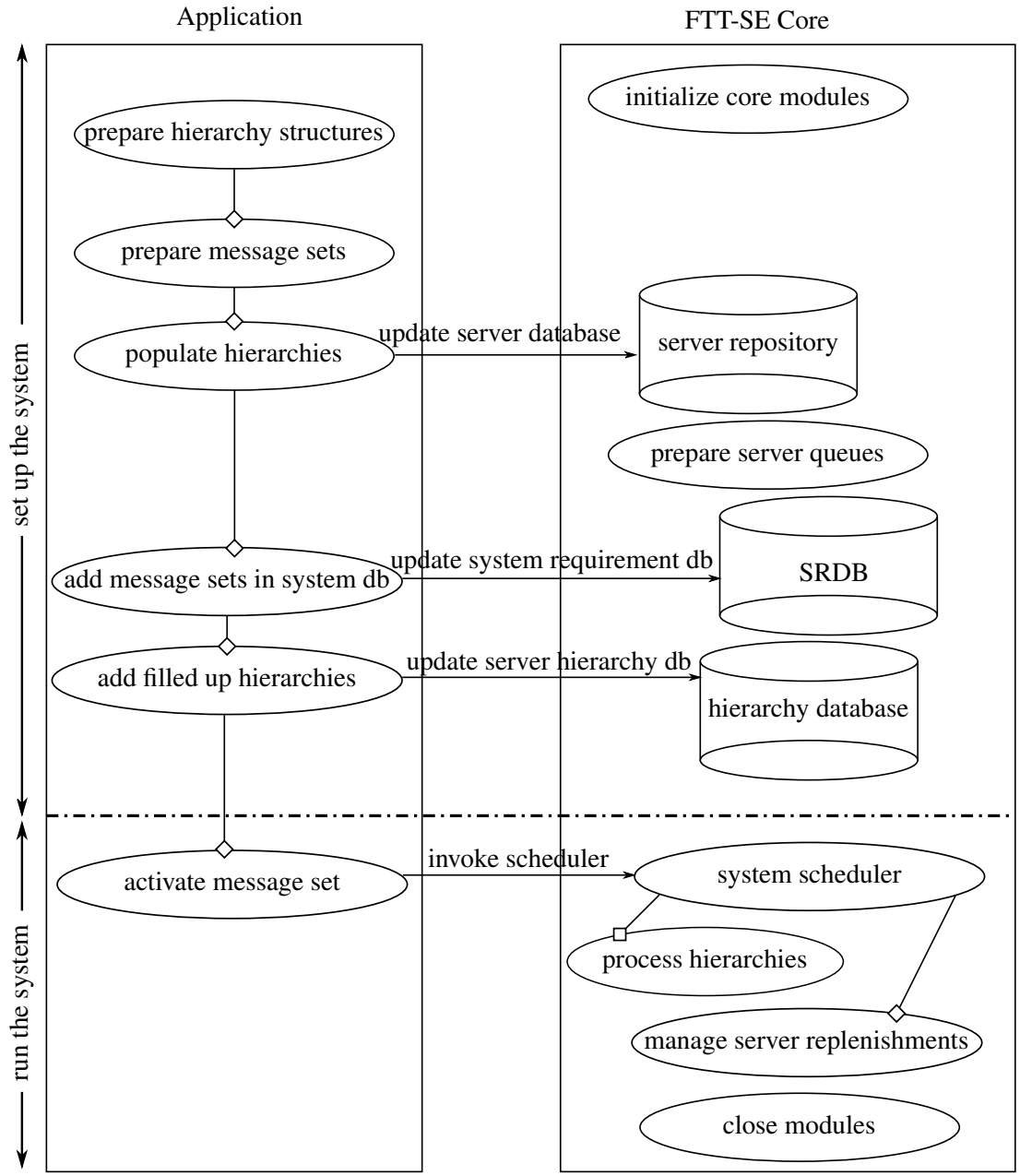


Figure 8.3: Application design and execution flow with HSF within FTT-SE

8.3 Summary

In this chapter, we presented essential elements of the experimental framework that generates large, random message sets and hierarchies. We briefly described the hierarchical server structures,

interface composition rules, message set generation and activation principles. We also showed how the experimental framework integrates with the FTT-SE core layer. Finally, this chapter showed the design flow required to build applications for implementing hierarchical server-based scheduling strategies as part of the FTT-SE master scheduler. In particular, we described the specific repositories and highlighted important interactions between the application program and the FTT-SE core to achieve this objective.

Chapter 9

Conclusion and Future Work

This chapter summarises the insights and findings of this thesis and outlines potential directions for further research on the topic.
CPS present an integrated view of systems made of tightly integrated, heterogeneous components consolidating control and communications with the physical environment. Such systems exceed in scope and potential today's embedded systems that are more focused on the computational platform, only. CPS rely on the support from several areas including, but not limited to, networks, control, security etc. Concerning the work presented in this thesis, we argued that the underlying communication network is an essential element to realise the objective of CPS. Thus, we investigated techniques that allow efficient use of the network resource. The following important considerations underline the importance of such communication design techniques.

- critical functions, frequently realised through distributed transactions on the network, must satisfy specific timing requirements
- applications may access the medium in several possible heterogeneous manners (e.g., short or long interarrival times, burstiness, higher or lower bandwidth requirements)
- dynamic behaviour of the system must be supported, i.e., failing links, changing application requirements

We proposed to address these challenges using the *resource reservation* techniques, enforced with server-based scheduling. This thesis set out to explore the topic of resource reservation in networks for complex and dynamic distributed embedded systems. We started with the scope of the work, the motivation, and the associated research work. In Chapter 2, we explored several real-time Ethernet-based technologies that are enablers of efficient communications for modern CPS. Following a qualitative comparison based on some fundamental aspects impacting on communications, we concluded that FTT-SE better fits our work. In Chapter 3, we surveyed common server-based scheduling techniques, their hierarchical composition and how they should be adapted/reconfigured to support dynamic Quality of Service. The other Chapters, from 4 to 7, include our main contributions that validate our thesis, namely the assessment of the efficiency of flat native reservations within FTT-SE, implementation of a hierarchical scheduling framework based on

polling and sporadic servers in the context of the FTT-SE, and a method to design the reservations considering polling servers. Chapter 8 includes a practical contribution to the design and execution of applications based on multi-level hierarchies of servers.

9.1 Thesis validation

In Chapter 1, we stated the thesis supported by this dissertation as follows:

> *The resource reservation paradigm is an effective means to segregate the communications from multiple heterogeneous applications in a distributed embedded system, potentially with mixed criticality levels, diverse real-time requirements and evolving configurations, thus supporting composability. In particular, we claim that this paradigm can be efficiently deployed over Ethernet, using the FTT-SE protocol, providing multiple levels of traffic isolation and constrained latency guarantees.*

We showed the validity of this claim in two parts. Firstly, we focused on simple sporadic reservations or flat servers. These servers control the burstiness in respective messages such that the inter-transmission time of the message is never less than the server period. Thus, these reservations shape the traffic submitted by individual streams, in that, other messages remain unaffected regarding their response times, whenever one or more messages arrive in bursts. This isolation was established in Chapter 4, together with an analytic worst-case delay model that was shown to be efficient in the sense that it captured the exact worst-case in a relatively large fraction of the messages in random sets, with residual cases of large deviations.

The study in Chapter 4 also provides the system designers with specific guidelines regarding the impact of the system configuration (either network, protocol or workload parameters) on the resource efficiency.

Concerning workload parameters, we changed message periods in two ways, experimenting with harmonic or prime periods, or using message set periods in different ranges.

For the first experiment, the analysis estimates were more accurate for messages with prime periods, i.e., on average, a match occurred in 40% messages of each prime set against 25% for harmonic. This is due to relative prime periods generating all possible relative offsets, as opposed to harmonic periods, thus exposing the true worst-case interference patterns.

Regarding the change in period range, we experimented using periods in three different ranges in our experiments, small $(5,10)$, medium $(5,60)$, and long $(5,500)$ and we observed that a short span of periods is bound to present higher matches to the analytic estimates, considering the schedulable sets. For the long range of periods, the actual absolute difference can be significant, though the percentual increase is not.

Concerning the protocol configuration parameters, we changed the size of the transmission window. We observed that when flat servers were scheduled within a larger partition, associated streams got short response times and the analysis reported better accuracy.

Finally, we experimented changing the available link utilization. In different experiments, we used

20%, 60% and 90% of the schedulable capacity of the transmission window. We observed that the analysis was more efficient for lower utilization of the window and becomes more pessimistic with growing link utilization.
Secondly, we show the validity of our thesis with hierarchical compositions of servers, based on the work reported in Chapters 5 and 6. This work achieved objective i), namely

i) *application-oriented semantics with a hierarchy, i.e., a complex application could be divided into sub-applications and reservations can be associated with applications with a single high-level interface but supporting internal nested reservations as requested by the applications;*

This work showed that a multi-level hierarchical server-based scheduling framework could be efficiently deployed using FTT-SE, with either the polling server policy (Chapter 5) or the sporadic server policy (Chapter 6). In particular, we presented the concept of an ISH (Independent Server Hierarchy) that arranges multiple servers in a hierarchy enabling bandwidth partitioning in multiple levels and the assignments of different bandwidth shares to different logical sub-parts of the application. Server-based scheduling policies then enforce temporal isolation between various applications or application sub-parts. We verified this property among message streams handled by different servers in case of overload and changes in the hierarchical architecture. The property of temporal isolation was tested in both cases, i.e., with polling or sporadic servers. Furthermore, sporadic servers reported shorter application response times, as expected, since these servers keep their capacity even when there are no pending requests, and thus can immediately serve the corresponding applications upon request. We further verified the property of temporal isolation among different applications by inducing congestion in different parts of randomly generated hierarchies and messages. Our results showed that the messages that behaved according to their periodic activation pattern remained unaffected by the induced burst. Random simulations verifying the temporal isolation property were carried out for the case of polling servers.
On the other hand, the claim on efficiency is supported on the work in Chapter 7, where we presented an approach for server design in an HSF. We considered the polling server policy and the non-preemptive nature of packet scheduling. The given method, essentially, creates server interfaces such that respective messages are scheduled very close to their deadlines. This approach is interesting for the bandwidth efficiency, as we saw that it required approximately half the combined utilization of all the hierarchies in the system when compared to another approach which we called the *naive approach* and which designs server interfaces focusing just on the message schedulability without any considerations for the bandwidth efficiency. The *naive approach* creates significantly larger reservations. We empirically tested the tightness of the server interfaces produced by our approach. We found that a decrease in server capacities as small as 1% saturated the system leading to strong deadline misses, thus pointing to a near-optimal HSF design that requests the least system resources for guaranteed schedulability. Considering that, whenever resources are available, the servers capacities can be relaxed to produce shorter response times, we claim that this work also meets objective ii), namely

ii) *flexible reservations, which provide a minimum guarantee and then offer more whenever there are available resources;*

Moreover, the framework could empirically test several use cases where we allocated more resources when the requested bandwidth was higher, tracking evolving requirements, or conversely, allocated different shares to different branches in the same hierarchy, thus, inherently supporting many different workload scenarios. With this level of flexibility we claim that our HSF on Ethernet with FTT-SE also meets objectives iii) and iv), namely:

iii) *a scalable solution so that it can work on networks with many flows associated with many and heterogeneous applications. For example, from industrial equipment to vehicles;*

iv) *a load-aware solution, allocating less additional resources on more loaded links, meeting end-to-end constraints, acting on deadlines, on priorities or even on the period/capacity of the reservations, according to what the underlying protocol allows;*

Altogether, the works presented in Chapters 4, 5, 6, and 7 showed that FTT-SE can indeed provide different types of network reservations with bounded latency and mutual isolation in a dynamically reconfigurable setting, as required by complex CPS that run multiple concurrent applications. Finally, Chapter 8 focused on the run-time mechanisms to support the hierarchical reservations, and on how these shall be used by a system designer, exposing the effectiveness of the implementation over FTT-SE.

9.2 Future Work

In the course of our investigation, we found several issues that would be worth studying in more detail but which could not be explored in this work. We report here some of those potential lines for future research:

- **Hierarchical resource reservations for multi-switch networks**
  The FTT-SE protocol was originally designed for a single switch case thereby limiting the size of the network. We carried out a work that focused on increasing the network size with multiple switches with a single FTT-SE master node that provided the transmission schedules for the entire network [43, 44]. In this architecture, we adapted the FTT-SE master scheduler to account for the various types of delays and path interferences arising due to the multi-switch case. This version of FTT-SE, however, lacked the support for hierarchical reservations. Adding such support in the multi-switch case is interesting since it expands the protocol applicability. But, there will be different aspects to consider. The ISH that we used in this work managed communication from one source to one destination. When the communication endpoints are on the same switch, it would be the same for the multi-switch case. However, when, source and destination are connected to different switches, then, there are one or more links between the switches where unrelated traffic would interfere. We can extend the concept of ISH such that one ISH will manage all applications that send data

and share the source, destination and hence the path. This would naturally increase the implementation complexity among other factors.

- **Analysis of sporadic servers** As we found in Chapter 4 for flat sporadic servers, there were some system configurations where analytic estimates were pessimistic. In general, though there were significant matches, it would be interesting to investigate further those corner cases where analysis was very pessimistic and refine the analysis. On the other hand, as noted in [37], the analysis of hierarchical sporadic servers was not addressed. Completing that analysis would be an interesting exercise.

- **On server design approaches** Server design approach that we presented comes with assumptions regarding strict periodicity of the root servers and works only for polling servers. We could investigate how to adapt the design approach when there might be jitter in the activations of different servers along the same path. Another extension of this work would be to investigate the adaptations needed for designing sporadic servers.

- **Connection to SDN** Software Defined Networks [144, 145] is an emerging paradigm according to which the network control is separated from the network data and centralised in so-called SDN controllers. We believe this paradigm bears many resemblances with FTT-SE but at a higher layer in the protocol stack. It would be interesting to study whether FTT-SE could be used to provide support to SDN, enforcing its controls locally.

Bibliography

[1] P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Springer Science & Business Media, 2011.

[2] G. C. Buttazzo, "Emerging Real-Time Methodologies," in *Embedded Computing Systems: Applications, Optimization, and Advanced Design*, pp. 140–159, IGI Global, Jan. 2013.

[3] E. A. Lee, "Cyber-Physical Systems - Are Computing Foundations Adequate," in *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, vol. 2, Oct. 2006.

[4] R. Baheti and H. Gill, "Cyber-physical Systems," *The impact of control technology*, vol. 12, pp. 161–166, 2011.

[5] P. Asare, D. Broman, E. A. Lee, G. Prinsloo, M. Torngren, and S. S. Sunder, "Cyber-Physical Systems." http://cyberphysicalsystems.org/. Accessed: 2015-10-05.

[6] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2011.

[7] C. R. Guareis de Farias, *Architectural Design of Groupware Systems: a Component-Based Approach*. Twente University Press, 2002.

[8] I. Crnkovic, "Component-based approach for embedded systems," in *9th International Workshop on Component-Oriented Programming (WCOP'2004)*, Jun 2004.

[9] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Power Optimization in Embedded Systems via Feedback Control of Resource Allocation," *IEEE Transactions on Control Systems Technology*, vol. 21, pp. 239–246, Jan. 2013.

[10] S. Shenker, D. D. Clark, and L. Zhang, "A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Network," *preprint*, 1993. https://www.researchgate.net/profile/Lixia_Zhang/publication/2810621_A_Scheduling_Service_Model_and_a_Scheduling_Architecture_for_an_Integrated_Services_Packet_Network/links/54eb3e1a0cf27a6de1176cb0.pdf.

[11] B. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, RFC Editor, June 1994.

[12] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, RFC Editor, Dec 1998.

[13] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, RFC Editor, Jan 2001.

[14] N. INSTRUMENTS, "Controller Area Network (CAN) Overview." http://www.ni.com/white-paper/2732/en/. Accessed: 2018-05-17.

[15] CiA, "History of CAN technology." https://www.can-cia.org/can-knowledge/can/can-history/. Accessed: 2018-05-19.

[16] "Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling," ISO 11898-1:2015, International Organization for Standardization, Dec 2015.

[17] N. INSTRUMENTS, "Introduction to the Local Interconnect Network (LIN) Bus." http://www.ni.com/white-paper/9733/en/. Accessed: 2018-05-17.

[18] "Road vehicles — Local Interconnect Network (LIN) — Part 1: General information and use case definition," ISO ISO 17987-1:2016, International Organization for Standardization, Aug 2016.

[19] N. INSTRUMENTS, "FlexRay Automotive Communication Bus Overview." http://www.ni.com/white-paper/3352/en/. Accessed: 2018-05-16.

[20] "Road vehicles – FlexRay communications system – Part 1: General information and use case definition," ISO 17458-1:2013, International Organization for Standardization, Feb 2013.

[21] L. L. Bello, "The case for Ethernet in Automotive Communications," *ACM SIGBED Review*, vol. 8, no. 4, pp. 7–15, 2011.

[22] "IEEE Standard for Local and metropolitan area networks–Audio Video Bridging (AVB) Systems," *IEEE Std 802.1BA-2011*, pp. 1–45, Sept. 2011.

[23] A. Specification, "7: Avionics Full Duplex Switched Ethernet (AFDX) Network," *ARINC Specification 664p7*, vol. 1, no. 2, p. 7, 2005.

[24] "Profinet Ethernet Standard." http://www.profibus.com. Accessed: 2017-11-21.

[25] TTTech, "Time-Triggered Ethernet." https://www.tttech.com/technologies/deterministic-ethernet/time-triggered-ethernet/, November 2017. Accessed: 2017-11-17.

[26] J. Strosnider, J. Lehoczky, and L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *IEEE Transactions on Computers*, vol. 44, pp. 73–91, Jan. 1995.

[27] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems," *Real-Time Systems*, vol. 1, pp. 27–60, Jun 1989.

[28] M. Spuri and G. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Real-Time Systems*, vol. 10, no. 2, pp. 179–210, 1996.

[29] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *19th IEEE International Real-Time Systems Symposium (RTSS 1998)*, pp. 4–13, Dec. 1998.

[30] L. Abeni and G. Buttazzo, "Resource Reservation in Dynamic Real-Time Systems," *Real-Time Systems*, vol. 27, pp. 123–167, July 2004.

[31] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing Real-Time Communication over COTS Ethernet switches," in *6th IEEE International Workshop on Factory Communication Systems (WFCS 2006)*, pp. 295–302, June 2006.

[32] H.-T. Lim, B. Krebs, L. Völker, and P. Zahrer, "Performance Evaluation of the Inter-Domain Communication in a Switched Ethernet Based In-Car Network," in *36th IEEE Conference on Local Computer Networks (LCN 2011)*, pp. 101–108, Oct. 2011.

[33] F. Simonot-Lion, "In car embedded electronic architectures: how to ensure their safety," in *5th IFAC International Conference on Fieldbus Systems and their Applications (FeT 2003)*, pp. 1–8, July 2003.

[34] L. Almeida, S. Fischmeister, M. Anand, and I. Lee, "A Dynamic Scheduling Approach to Designing Flexible Safety-critical Systems," in *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT '07)*, (New York, NY, USA), pp. 67–74, ACM, 2007.

[35] "Flexible Time-Triggered Communications." www.fe.up.pt/ftt. [Online; accessed 19-June-2013].

[36] Z. Iqbal, L. Almeida, and M. Ashjaei, "Analyzing the Efficiency of Sporadic Reservations on Ethernet with FTT-SE," in *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2017)*, pp. 1–8, Sept 2017.

[37] Z. Iqbal and L. Almeida, "Towards an analysis for hierarchies of sporadic servers on Ethernet," in *11th IEEE Symposium on Industrial Embedded Systems (SIES 2016)*, pp. 1–6, May 2016.

[38] Z. Iqbal, L. Almeida, and M. Behnam, "Efficiency study for sporadic servers on Ethernet with FTT-SE," in *9th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2016) in conjunction with the 37th IEEE International Real-Time Systems Symposium (RTSS 2016)*, pp. 25–26, November 2016.

[39] Z. Iqbal, L. Almeida, M. Ashjaei, and M. Behnam, "On the efficiency of sporadic servers on Ethernet with FTT-SE," *SIGBED Rev.*, vol. 14, pp. 32–34, Nov. 2017.

[40] Z. Iqbal, L. Almeida, R. Marau, M. Behnam, and T. Nolte, "Implementing Hierarchical Scheduling on COTS Ethernet Switches Using a Master/Slave Approach," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES 2012)*, pp. 76–84, June 2012.

[41] Z. Iqbal, L. Almeida, and M. Behnam, "Implementing Virtual Channels in Ethernet using Hierarchical Sporadic Servers," in *The 12th International Workshop on Real-Time Networks (RTN 2013) in conjunction with the 25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*, July 2013.

[42] Z. Iqbal, L. Almeida, and M. Behnam, "Designing Network Servers within a Hierarchical Scheduling Framework," in *30th Annual ACM Symposium on Applied Computing (SAC 2015)*, pp. 653–658, Apr. 2015.

[43] M. Behnam, Z. Iqbal, P. Silva, R. Marau, L. Almeida, and P. Portugal, "Engineering and Analyzing Multi-Switch Networks with Single Point of Control," in *1st International Workshop on Worst-Case Traversal Time (WCTT'11) in conjunction with the 32nd IEEE International Real-Time Systems Symposium (RTSS'11)*, pp. 11–18, Nov. 2011.

[44] R. Marau, M. Behnam, Z. Iqbal, P. Silva, L. Almeida, and P. Portugal, "Controlling Multi-Switch Networks for Prompt Reconfiguration," in *9th IEEE International Workshop on Factory Communication Systems (WFCS 2012)*, pp. 233–242, May 2012.

[45] A. Y. Chong and C. S. Chua, *Driving Asia: As Automotive Electronic Transforms a Region*. Infineon Technologies Asia Pacific Pte Limited, 2011.

[46] Statista, "Automotive electronics cost as a percentage of total car cost worldwide from 1950 to 2030." https://www.statista.com/statistics/277931/automotive-electronics-cost-as-a-share-of-total-car-cost-worldwide/. Accessed: 2018-03-04.

[47] C. Mathas, "The price tag of automotive electronics: What's really at play?." https://www.edn.com/electronics-blogs/engineering-on-wheels/4458881/The-price-tag-of-automotive-electronics--What-s-really-at-play-. Accessed: 2018-03-04.

[48] E. Christmann, "Data Communication in the Automobile - Part 1." https://elearning.vector.com/portal/medien/cmc/press/PTR/SerialBusSystems_Part1_ElektronikAutomotive_200611_PressArticle_EN.pdf. Accessed: 2015-09-20.

[49] E. Christmann, "Data Communication in the Automobile - Part 2." https://elearning.vector.com/portal/medien/cmc/press/PTR/SerialBusSystems_Part2_ElektronikAutomotive_200612_PressArticle_EN.pdf. Accessed 2015-09-30.

[50] A. Sangiovanni-Vincentelli and M. D. Natale, "Embedded System Design for Automotive Applications," *Computer*, vol. 40, pp. 42–51, Oct. 2007.

[51] O. Scheickl, C. Ainhauser, and M. Rudorfer, "Distributed Development of Automotive Real-time Systems based on Function-triggered Timing Constraints," in *Embedded Real-time Software and Systems (ERTS2 2010)*, May 2010.

[52] R. Obermaisser, B. Frömel, C. El Salloum, and B. Huber, "Integrating Safety and Multimedia Subsystems on a Time-Triggered System-on-a-Chip," in *6th IEEE International Conference on Industrial Informatics (INDIN 2008)*, pp. 270–275, July 2008.

[53] T. Nolte, H. Hansson, and L. L. Bello, "Automotive Communications-Past, Current and Future," in *10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, pp. 985–992, Sept. 2005.

[54] T. Steinbach, F. Korf, and T. C. Schmidt, "Real-time Ethernet for Automotive Applications: A Solution for Future In-Car Networks," in *1st IEEE International Conference on Consumer Electronics-Berlin (ICCE-Berlin 2011)*, pp. 216–220, Sept. 2011.

[55] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication," in *40th IEEE Conference on Local Computer Networks (LCN 2015)*, pp. 1–9, Oct. 2015.

[56] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *1st international conference on simulation tools and techniques for communications, networks and systems & workshops (Simutools08)*, p. 60, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Mar. 2008.

[57] M. Behnam, R. Marau, and P. Pedreiras, "Analysis and Optimization of the MTU in Real-Time Communications over Switched Ethernet," in *16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011)*, pp. 1–7, Sept. 2011.

[58] "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, Mar. 2016.

[59] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)," in *76th IEEE Vehicular Technology Conference (VTC Fall 2012)*, pp. 1–5, Sept. 2012.

[60] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, vol. 19, pp. 395–404, July 1976.

[61] J. Jasperneite and P. Neumann, "Switched Ethernet for factory communication," in *8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2001)*, pp. 205–212, Oct. 2001.

[62] AEE Committee *et al.*, "Aircraft Data Network Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network, ARINC Specification 664," *Annapolis, Maryland: Aeronautical Radio*, 2002.

[63] M. LLC, "AFDX/ARINC 664 Protocol Tutorial." http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm, 1999. Accessed: 2017-10-5.

[64] "IEEE Draft Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," *IEEE Std P802.1AS/D7.7*, pp. 1–296, Nov. 2010.

[65] "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)," *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)*, pp. 1–119, Sept. 2010.

[66] "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. C1–72, Jan. 2009.

[67] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, pp. 558–565, July 1978.

[68] K. Merkel, "Hybrid Broadcast Broadband TV, The New Way to a Comprehensive TV Experience," in *14th ITG Conference on Electronic Media Technology (CEMT 2011)*, pp. 1–4, Mar. 2011.

[69] "Example of Automotive Multimedia Test: Multiple Display Synchronization." http://www.ni.com/white-paper/14369/en/. Accessed: 2017-10-29.

[70] ITU-R, "Relative timing of sound and vision for broadcasting," Recommendation BT.1359-1, International Telecommunication Union, Nov. 1998.

[71] M. O. van Deventer, H. Stokking, M. Hammond, J. Le Feuvre, and P. Cesar, "Standards for multi-stream and multi-device media synchronization," *IEEE Communications Magazine*, vol. 54, pp. 16–21, Mar. 2016.

[72] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. 1–300, July 2008.

[73] W. Steiner, N. Finn, and M. Posch, "IEEE 802.1 Audio/Video Bridging and Time-Sensitive Networking," in *Industrial Communication Technology Handbook, Second Edition*, ch. 20, Oxford: CRC Press, 2017.

[74] B. SYSTEMS, "AVB RESOURCE GUIDE." http://c353616.r16.cf1.rackcdn.com/Biamp_AVB_Reference_Guide_Apr14.pdf. Accessed: 2018-01-20.

[75] M. Glaß, S. Graf, F. Reimann, and J. Teich, "Design and Evaluation of Future Ethernet AVB-Based ECU Networks," in *Embedded Systems Development*, vol. 20, ch. 12, pp. 205–220, Springer, Jul 2014.

[76] J. Cao, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, "Tight Worst-Case Response-Time Analysis for Ethernet AVB using Eligible Intervals," in *12th IEEE World Conference on Factory Communication Systems (WFCS 2016)*, pp. 1–8, May 2016.

[77] J. Cao, P. J. Cuijpers, R. J. Bril, and J. J. Lukkien, "Independent yet Tight WCRT Analysis for Individual Priority Classes in Ethernet AVB," in *24th International Conference on Real-Time Networks and Systems (RTNS 2016)*, pp. 55–64, Oct. 2016.

[78] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng, "Schedulability Analysis of Ethernet AVB Switches," in *20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2014)*, pp. 1–10, Aug. 2014.

[79] M. D. J. Teener, A. N. Fredette, C. Boiger, P. Klein, C. Gunther, D. Olsen, and K. Stanton, "Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging," *Proceedings of the IEEE*, vol. 101, pp. 2339–2354, Nov. 2013.

[80] avnu.org, "Types of Traffic in AVB." http://avnu.org/wp-content/uploads/2014/05/AVnu-AAA2C_Types-of-Traffic-in-AVB-2_Michael-Johas-Teener-Markus-Jochim.pdf. Accessed: 2018-01-30.

[81] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design Optimization of TTEthernet-based Distributed Real-Time Systems," *Real-Time Systems*, vol. 51, no. 1, pp. 1–35, 2015.

[82] W. Steiner and B. Dutertre, "Automated Formal Verification of the TTEthernet Synchronization Quality," in *3rd International Conference on NASA Formal Methods (NFM'11)*, pp. 375–390, Apr. 2011.

[83] D. Tamas-Selicean, P. Pop, and W. Steiner, "Synthesis of Communication Schedules for TTEthernet-Based Mixed-Criticality Systems," in *8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '12)*, (New York, NY, USA), pp. 473–482, Oct. 2012.

[84] M. Boyer, H. Daigmorte, N. Navet, and J. Migge, "Performance impact of the interactions between time-triggered and rate-constrained transmissions in TTEthernet," in *8th European Congress on Embedded Real Time Software and Systems (ERTS2 2016)*, Jan. 2016.

[85] R. Marau, P. Pedreiras, and L. Almeida, "Asynchronous Traffic Signaling over Master-Slave Switched Ethernet protocols," in *6th International Workshop on Real Time Networks (RTN 2007)*, July 2007.

[86] R. Marau, L. Almeida, K. Lakshmanan, and R. Rajkumar, "Utilization-based Schedulability Analysis for Switched Ethernet aiming Dynamic QoS Management," in *15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2010)*, Sept. 2010.

[87] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Applying Trajectory approach with static priority queuing for improving the use of available AFDX resources," *Real-Time Systems*, vol. 48, pp. 101–133, Jan 2012.

[88] T. Hamza, J.-L. Scharbarg, and C. Fraboul, "QoS-aware AFDX: benefits of an efficient priority assignment for avionics flows," in *34th IEEE International Real-Time Systems Symposium: Work-in-progress session (RTSS 2013)*, pp. 13–14, Dec. 2013. Available at http://2013.ieee-rtss.org/wp-content/uploads/2013/11/WiP-proceedings.pdf, Accessed: 2018-01-04.

[89] L. L. Bello, "Novel Trends in Automotive Networks: A Perspective on Ethernet and the IEEE Audio Video Bridging," in *19th IEEE International Conference on Emerging Technology and Factory Automation (ETFA 2014)*, pp. 1–8, Sept. 2014.

[90] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan, "TTEthernet Dataflow Concept," in *8th IEEE International Symposium on Network Computing and Applications (NCA 2009)*, pp. 319–322, July 2009.

[91] D. TamasSelicean, P. Pop, and W. Steiner, "Timing Analysis of Rate Constrained Traffic for the TTEthernet Communication Protocol," in *18th IEEE International Symposium on Real-Time Distributed Computing (ISORC 2015)*, pp. 119–126, April 2015.

[92] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1qbv Time Sensitive Networks," in *24th International Conference on Real-Time Networks and Systems (RTNS 2016)*, RTNS '16, (New York, NY, USA), pp. 183–192, ACM, Oct. 2016.

[93] S. AS6802, "Sae standards," Nov. 2011.

[94] S. Kehrer, O. Kleineberg, and D. Heffernan, "A comparison of Fault-Tolerance Concepts for IEEE 802.1 Time Sensitive Networks (tsn)," in *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014)*, pp. 1–8, Sept. 2014.

[95] D. Gessner, *Adding Fault Tolerance to a Flexible Real-Time Ethernet Network for Embedded Systems*. PhD thesis, Universitat de les Illes Balears, 2017.

[96] J. P. Arenas, "FT4FTT — FT4FTT-Ethernet: Fault Tolerance mechanisms for adaptive distributed embedded systems based on FTT-Ethernet." http://srv.uib.es/ft4ftt/. Accessed: 2018-01-20.

[97] J. P. Arenas, "DFT4FTT - Dynamic Fault Tolerance for increasing the adaptivity of highly-reliable distributed embedded systems based on Flexible Time-Triggered Ethernet." http://srv.uib.es/dft4ftt/. Accessed: 2018-03-04.

[98] P. Pedreiras, "HaRTES - Hard Real-Time Ethernet Switching." http://hartes.av.it.pt/project.html, 2009. Accessed: 2017-11-21.

[99] R. Santos, A. Vieria, R. Marau, P. Pedreiras, A. Oliveira, L. Almeida, and T. Nolte, "Implementing Server-Based Communication within Ethernet Switches," in *2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'09) in conjunction with the 30th IEEE International Real-Time Systems Symposium (RTSS'09)*, Dec 2009.

[100] L. Pearson, "Stream Reservation Protocol - Revision 1.0." http://avnu.org/wp-content/uploads/2014/05/AVnu_Stream-Reservation-Protocol-v1.pdf. Accessed: 2018-01-20.

[101] "Deterministic Ethernet and Unified Networking." http://deterministic-ethernet.blogspot.pt/2011/06/why-deterministic-why-ethernet.html. Accessed: 2017-12-20.

[102] I. Álvarez, L. Almeida, and J. Proenza, "A First Qualitative Comparison of the Admission Control in FTT-SE, HaRTES and AVB," in *12th IEEE International Workshop on Factory Communication Systems (WFCS 2016)*, pp. 1–4, 2016.

[103] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM (JACM)*, vol. 20, pp. 46–61, Jan. 1973.

[104] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 3rd ed., 2011.

[105] M. Stanovich, T. P. Baker, A.-I. Wang, and M. G. Harbour, "Defects of the POSIX Sporadic Server and How to Correct Them," in *16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2010)*, pp. 35–45, Apr. 2010.

[106] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "HARD REAL-TIME SCHEDULING: THE DEADLINE-MONOTONIC APPROACH," *IFAC Proceedings Volumes*, vol. 24, no. 2, pp. 127–132, 1991.

[107] D. L. Sun, Z. Deng, and L. J. S. JW-s, "Dynamic Scheduling of Hard Real-Time Applications in Open System Environment," in *17th IEEE Real-Time Systems Symposium (RTSS'96)*, Dec. 1996.

[108] Z. Deng and J. W.-S. Liu, "Scheduling Real-Time Applications in an Open Environment," in *18th IEEE Real-Time Systems Symposium (RTSS'97)*, pp. 308–319, Dec. 1997.

[109] J. Löser and H. Härtig, "Low-latency Hard Real-Time Communication over Switched Ethernet," in *16th EUROMICRO Conference on Real-Time Systems (ECRTS 2004)*, pp. 13–22, July 2004.

[110] L. Almeida and P. Pedreiras, "Scheduling within Temporal Partitions: Response-time Analysis and Server Design," in *4th ACM International Conference on Embedded Software (EMSOFT 2004)*, pp. 95–103, Sept. 2004.

[111] G. Lipari and E. Bini, "Resource Partitioning among Real-Time Applications," in *15th Euromicro Conference on Real-Time Systems (ECRTS 2003)*, pp. 151–158, Jul 2003.

[112] I. Shin and I. Lee, "Periodic Resource Model for Compositional Real-Time Guarantees," in *24th IEEE International Real-Time Systems Symposium (RTSS'03)*, pp. 2–13, Dec. 2003.

[113] R. Davis and A. Burns, "An Investigation into Server Parameter Selection for Hierarchical Fixed Priority Pre-emptive Systems," in *16th International Conference on Real-Time and Network Systems (RTNS 2008)*, pp. 19–28, Oct. 2008.

[114] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems," in *7th ACM & IEEE International Conference on Embedded Software (EMSOFT 2007)*, pp. 279–288, Oct. 2007.

[115] M. Ashjaei, N. Khalilzad, S. Mubeen, M. Behnam, I. Sander, L. Almeida, and T. Nolte, "Designing end-to-end resource reservations in predictable distributed embedded systems," *Real-Time Systems*, vol. 53, pp. 916–956, Nov 2017.

[116] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee, "Incremental Schedulability Analysis of Hierarchical Real-Time Components," in *6th ACM & IEEE International Conference on Embedded Software (EMSOFT 2006)*, pp. 272–281, Oct. 2006.

[117] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," in *18th IEEE International Real-Time Systems Symposium (RTSS 1997)*, pp. 298–307, Dec. 1997.

[118] S. Ghosh, J. Hansen, R. Rajkumar, and J. Lehoczky, "Integrated Resource Management and Scheduling with Multi-Resource Constraints," in *25th IEEE International Real-Time Systems Symposium (RTSS 2004)*, pp. 12–22, Dec. 2004.

[119] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-allocation," in *7th IEEE International Workshop on Quality of Service (IWQoS'99 - 1999)*, pp. 27–36, 31 May–4 June 1999.

[120] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network: The Magazine of Global Internetworking*, vol. 7, pp. 8–18, Sept. 1993.

[121] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The Time-Triggered Ethernet (TTE) Design," in *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, pp. 22–33, May 2005.

[122] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, "Ttethernet: Time-triggered ethernet," 2011.

[123] M. A. Brown, "Linux Traffic Control." http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html.

[124] S. Varadarajan and T. Chiueh, "EtheReal: a host-transparent real-time Fast Ethernet switch," in *6th International Conference on Network Protocols*, pp. 12–21, Oct. 1998.

[125] H. Hoang, M. Jonsson, A. Kallerdahl, and U. Hagström, "Switched Real-Time Ethernet with Earliest Deadline First Scheduling—Protocols, Traffic Handling and Simulation Analysis," *Parallel and Distributed Computing Practices*, vol. 5, pp. 105–115, Mar. 2002.

[126] M. Zhang, J. Shi, T. Zhang, and Y. Hu, "Hard Real-time Communication over Multi-hop Switched Ethernet," in *3rd IEEE International Conference on Networking, Architecture, and Storage (NAS 2008)*, pp. 121–128, June 2008.

[127] G. Carvajal, M. Figueroa, R. Trausmuth, and S. Fischmeister, "*Atacama*: An Open FPGA-based Platform for Mixed-Criticality Communication in Multi-Segmented Ethernet Networks," in *21st IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM 2013)*, pp. 121–128, Apr. 2013.

[128] R. Marau, N. Figueiredo, R. Santos, P. Pedreiras, L. Almeida, and T. Nolte, "Server-based Real-Time Communications on Switched Ethernet," in *1st Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'08) in conjunction with the 29th IEEE International Real-Time Systems Symposium (RTSS'08)*, Nov 2008.

[129] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida, "Multi-level Hierarchical Scheduling in Ethernet Switches," in *11th ACM & IEEE International Conference on Embedded Software (EMSOFT'11)*, pp. 185–194, Oct. 2011.

[130] "Serv-CPS: Server-based Real-Time Ethernet Communication Architecture for Cyber-Physical Systems." http://serv-cps.av.it.pt/, 2012. Accessed: 2017-04-18.

[131] F. Yekeh, M. Pordel, L. Almeida, M. Behnam, and P. Portugal, "Exploring Alternatives to Scale FTT-SE to Large Networks," in *6th IEEE International Symposium on Industrial Embedded Systems (SIES 2011)*, pp. 107–110, June 2011.

[132] P. Pedreiras and L. Almeida, "Message Routing in Multi-Segment FTT Networks: The Isochronous Approach," in *18th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2004)*, pp. 122–129, Apr. 2004.

[133] M. Ashjaei, M. Liu, M. Behnam, A. Mifdaoui, L. Almeida, and T. Nolte, "Worst-Case Delay Analysis of Master-Slave Switched Ethernet Networks," in *2nd International Workshop on Worst-Case Traversal Time (WCTT'12) in conjunction with the 33rd IEEE International Real-Time Systems Symposium (RTSS'12)*, Dec. 2012.

[134] M. Ashjaei, P. Pedreiras, M. Behnam, R. J. Bril, L. Almeida, and T. Nolte, "Response Time Analysis of Multi-Hop HaRTES Ethernet Switch Networks," in *10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, pp. 1–10, May 2014.

[135] M. Ashjaei, M. Behnam, P. Pedreiras, R. J. Bril, L. Almeida, and T. Nolte, "Reduced Buffering Solution for Multi-Hop HaRTES Switched Ethernet Networks," in *20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2014)*, pp. 1–10, Aug. 2014.

[136] N. H. Weiderman and N. I. Kamenoff, "Hartstone Uniprocessor Benchmark: Definitions and Experiments for Real-Time Systems," *Real-Time Systems*, vol. 4, no. 4, pp. 353–382, 1992.

[137] R. R. D. Marau, *Real-time communications over switched Ethernet supporting dynamic QoS management*. Doutoramento em engenharia informática, Universidade de Aveiro, 2009.

[138] L. Almeida and J. A. Fonseca, "Analysis of a Simple Model for Non-Preemptive Blocking-Free Scheduling," in *13th Euromicro Conference on Real-Time Systems (ECRTS 2001)*, June 2001.

[139] E. Bini and G. C. Buttazzo, "Measuring the Performance of Schedulability Tests," *Real-Time Systems*, vol. 30, pp. 129–154, May 2005.

[140] D. Seto, J. P. Lehoczky, and L. Sha, "Task Period Selection and Schedulability in Real-Time Systems," in *19th IEEE International Real-Time Systems Symposium (RTSS 1998)*, pp. 188–198, IEEE, Dec. 1998.

[141] T. Chantem, X. Wang, M. D. Lemmon, and X. S. Hu, "Period and Deadline Selection for Schedulability in Real-Time Systems," in *20th Euromicro Conference on Real-Time Systems (ECRTS 2008)*, pp. 168–177, IEEE, July 2008.

[142] A. Easwaran, M. Anand, and I. Lee, "Compositional Analysis Framework using EDP Resource Models," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 129–138, Dec 2007.

[143] A. Easwaran, I. Lee, I. Shin, and O. Sokolsky, "Compositional Schedulability Analysis of Hierarchical Real-Time Systems," in *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pp. 274–281, May 2007.

[144] "OPEN NETWORKING FOUNDATION." https://www.opennetworking.org/index.php. Accessed: 2015-10-18.

[145] O. M. E. Committee *et al.*, "Software-Defined Networking: The New Norm for Networks," *ONF White Paper*, Apr 2012.